НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

«**До захисту допущено**»
В.о. завідувача кафедри

_____
__*О.А.Павлов*__
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки *6.050103 «Програмна інженерія»*

спеціальність *«Програмне забезпечення систем»*

на тему: *Мережеве ігрове програмне забезпечення на основі геолокації*

Виконав: студентка 4 курсу, групи ІП-51

Старченко Анастасія Сергіївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *асистент Ісаченко Г.В.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *ст. викл. Головченко М.М.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5120.045490.01.91	Технічне завдання	11	
3	А4	КПІ.ІП-5120.045490.02.81	Пояснювальна записка	69	
4	А4	КПІ.ІП-5120.045490.03.13	Опис програми	14	
5	А4	КПІ.ІП-5120.045490.04.51	Програма та методика тестування	10	
6	А4	КПІ.ІП-5120.045490.05.34	Керівництво користувача	7	
7	А3	КПІ.ІП-5120.045490.06.99 КЕ	Креслення вигляду екранних форм	1	
8	А3	КПІ.ІП-5120.045490.06.99 СС	Схема структурна діяльності атаки	1	
9	А3	КПІ.ІП-5120.045490.06.99 СС	Схема структурна класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-5120.045490.00.90		
Розробн.		Старченко А.С.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Ісаченко Г.В.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-51	
Н/контр.		Головченко М.М.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 16 таблиць та 12 джерел – загалом 69 сторінок.

Об`єкт дослідження: мережеве ігрове програмне забезпечення з використанням геолокації.

Мета дипломного проекту: розробити гру на основі геолокацію, зробити її простішою, ніж у конкурентів, підвищити користь від комп'ютерних ігор, мотивуючи гравців гуляти пішки.

У першому розділі наведено аналіз предментої області, аналіз відомих технічних рішень та аналіз потенційних конкурентів. На основі дослідження сформовано вимоги до проекту.

Другий розділ присвячений моделюванню та конструюванню програмного забезпечення, розробці архітектури, інтерфейсів, классів, бази даних.

Третій розділ містить опис аналізу якості програми та загальний опис етапів та процесів тестування.

У четвертому розділі описано інструменту щодо розгортання та супровіду даного програмного забезпечення.

Також наведено: опис програми, інструкцію користувача, програму та методику тестування, креслення вигляду екранних форм, схему структурну діяльності, схему структурну класів програмного забезпечення.

КЛЮЧОВІ СЛОВА: ГЕОЛОКАЦІЯ, ГРА, IOS, SWIFT, CORELOCATION, MAPKIT

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 16 tables and 12 sources - a total of 69 pages.

Object of research: network game software based on geolocation.

The purpose of the diploma project: to develop a game based on geolocation, make it easier than existing games, increase the benefits of computer games, motivating players to walk.

The first section provides an analysis of the industry, analysis of existing technical solutions and analysis of potential competitors. On the basis of research the requirements for the project have been formed.

The second section is devoted to modeling and designing, developing architecture, interfaces, classes, database.

The third section contains a description of the quality analysis of the program and a general description of the stages and processes of testing.

The fourth section describes the deployment and maintenance tools for this software.

Also provided: description of the program, user's manual, testing program and methods, drawing of the appearance of the screen forms, activity structure scheme, software classes structure scheme.

KEYWORDS: GEOLOCATION, GAME, IOS, SWIFT, CORELOCATION, MAPKIT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Геймплей або ігровий процес (англ. Gameplay) – компонент гри, що відповідає за інтерактивну взаємодію гри та гравця. Геймплей описує, як гравець взаємодіє з ігровим світом, як ігровий світ реагує на дії гравця та як визначається набір дій, що пропонує гравцю гра.

UX (user experience) – досвід взаємодії; сприйняття та дії користувача, що виникають в результаті використання продукції, системи або послуги.

ВСТУП

До 21-го сфера століття розваг обмежувалась щонайменше законами фізики. Так, ми могли створити найгарніших та найреалістичніших ляльок, та ми не могли змусити їх оживати. Ми могли уявляти себе чарівниками, граючи с друзями на подвір'ї, але наврядче комусь з нас насправді вдавалося начаклувати вогонь чи злетіти до хмар. Магія переважно відбувалась у нашій уяві. Але зараз вся наша уява перенеслась в екрани телефонів, і ми можемо дійсно побачити всі ті дива, що раніше були тільки в голові.

Технології віртуальної та доповненої реальності зробили ігровий процес неймовірно реалістичним. Тепер магія неначе втручається в реальний світ та стає його частиною. Сьогодні розваги більше не обмежуються законами фізики. Єдиним обмеженням є наша уява.

Більшість розробників використовує дуже обмежений список можливостей сучасного технологічного потенціалу. Набагато простіше залишатись у зоні комфорту, і робити звичні речі, просто адаптуючи їх для смартфонів – кросворди, ігри "щоб вбити час", симулятори популярних настолок – саме таких додатків більше 90 відсотків на ринку. Але є й такі, що мислять "поза коробкою", що намагаються створити те, що стало можливо тільки теперь, щось принципове нове.

Сьогодні як ніколи просто зробити наступний крок у розвитку ігор – перенести їх з екранів у життя. Метою цієї роботи було саме зробити щось принципове нове у сфері розваг, розмити кордон між уявою та реальністю, задіявши мобільні технології, а саме – технологію геолокації.

Важлива функція таких додатків – мотивація для людей виходити з дому та пересуватись своїм містом. Для цього розробники розміщають важливі ігрові об'єкти поруч із фонтанами, пам'ятками, визначними будівлями. Тоді гра суміщається з приємною та корисною прогулянкою.

Мета даної розробки: створити мотивуючу гру на основі геолокації, врахувавши помилки попередників, підвищити мотивуючу функцію.

Результатом роботи став мобільний додаток, у якому ігровий процес переноситься у реальний світ, а від учасників вимагається пересування у справжньому, не уявному чи намальованому, просторі. Інтеграція ігрового досвіду в життя – саме та перевага, якої не було у нас до появи мобільних телефонів. Ця гра використовує її на повну.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

За одним із визначень, гра – це вид непродуктивної діяльності, де мотив полягає не в її результаті, а власне у процесі. Також грою називають інструменти (наприклад, картки чи програма), що призначені для такої діяльності. За іншим визначенням, гра – це діяльність, що полягає в досягненні цілей (наприклад, через подолання перешкод) з метою отримання задоволення (відмінно від роботи, де головна ціль зазвичай – матеріальна нагорода). Тобто, невід'ємними складовими будь-якої гри є певні цілі та цікавий процес їх досягнення.

Геолокація – це визначення місця положення об'єкта в реальному світі, наприклад, мобільного телефона чи радару. У найпростішому випадку, результатом геолокації є множина географічних координат. Зазвичай, ці координати набувають змісту при перетворені на певну зрозумілу інформацію – наприклад, назву вулиці.

Для геолокації пристрій може використовувати навігацію за допомогою супутників (відому як GPS). Коли доступ до супутника неможливий, може використовуватися інформація з веж сотового зв'язку – за допомогою метода триангуляції можна вирахувати приблизну позицію. Цей метод не є таким точним, як GPS, проте в останні роки його точність швидко покращується.

Також інформація про локацію девайса може бути здобута через вираховування географічного положення через IP-адресу, MAC адресу, Wi-Fi, так званий "відбиток девайсу", тощо. Такими методами користуються зазвичай, коли людина не дає згоди поширення свого місця знаходження. Цим можуть користуватись шахраї, проте багато міжнародних організацій фінансують такі "дедуктивні" методи геолокації для боротьби з тероризмом.

У більшості випадків, легально отримати локацію людини можна тільки з її дозволу [1].

Принцип геолокації зображено на рисунку 1.1.

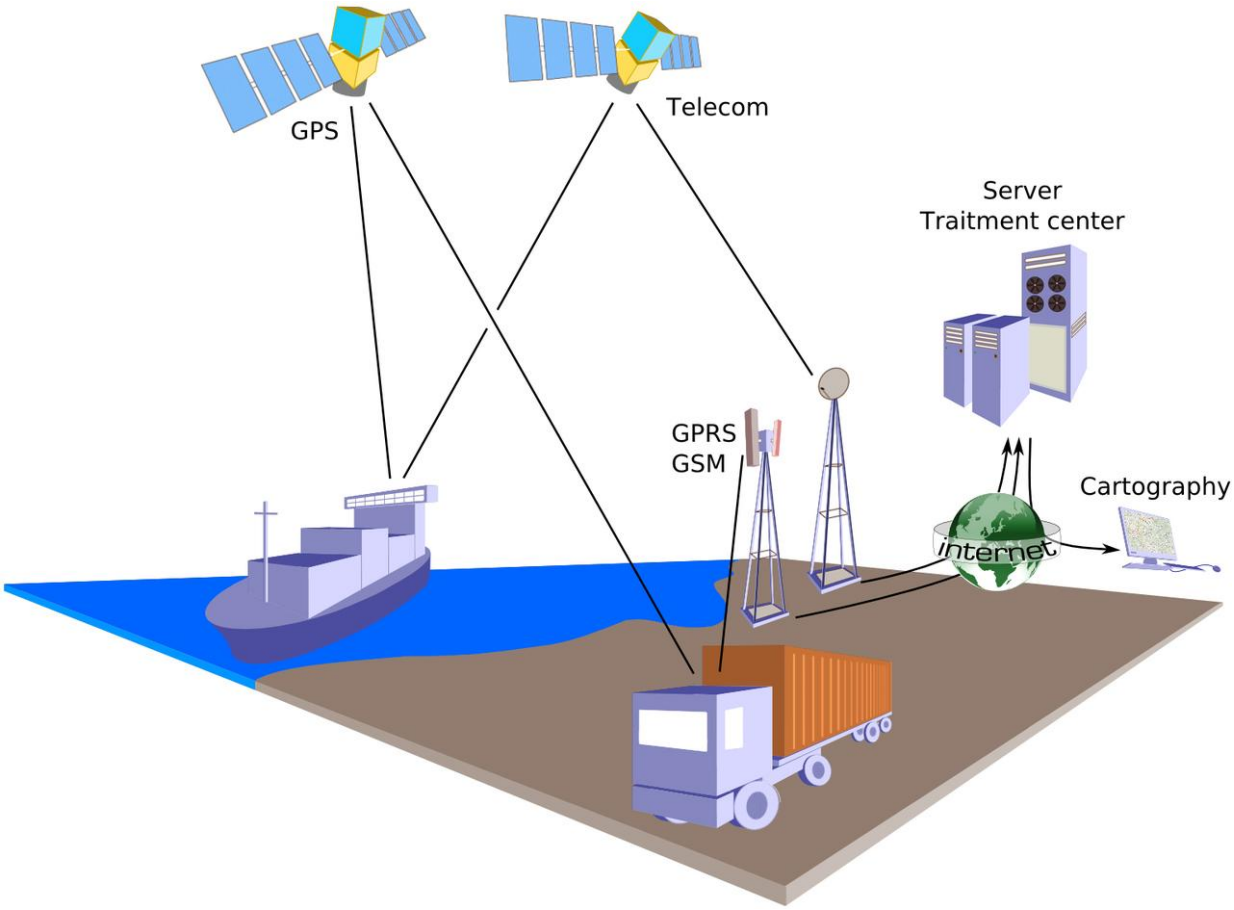

Рисунок 1.1 - Принцип геолокації

Термін “геолокація” також означає координати довжини та широти певного місця. Саме це використання наразі є найбільш вживаним.

Поєднання ігор та геолокації є відносно новою ідеєю – перші такі проекти були створені в 2012 році. Тоді вони не здобули значної популярності з низки причин, головна з яких – обмеженність в технологіях на той час. За останні ж кілька років кілька величезних корпорацій (як то Apple, Google та інші) інвестували чимало коштів у розвиток індустрії. Результатом стало винекнення технології доповненої реальності, AR (Augumented Reality).

Формула геолокація + гра + AR виявилася надцікавою для користувачів та розробників, і вже скоро ми побачили Pokemon Go – наразі відому чи не кожному власнику смартфона багатокористувацьку гру, метою якої було знаходити покемонів (магічних віртуальних створінь) в реальному світі, ловити їх та тренувати. Ігровий процес зображено на рисунку 1.2.

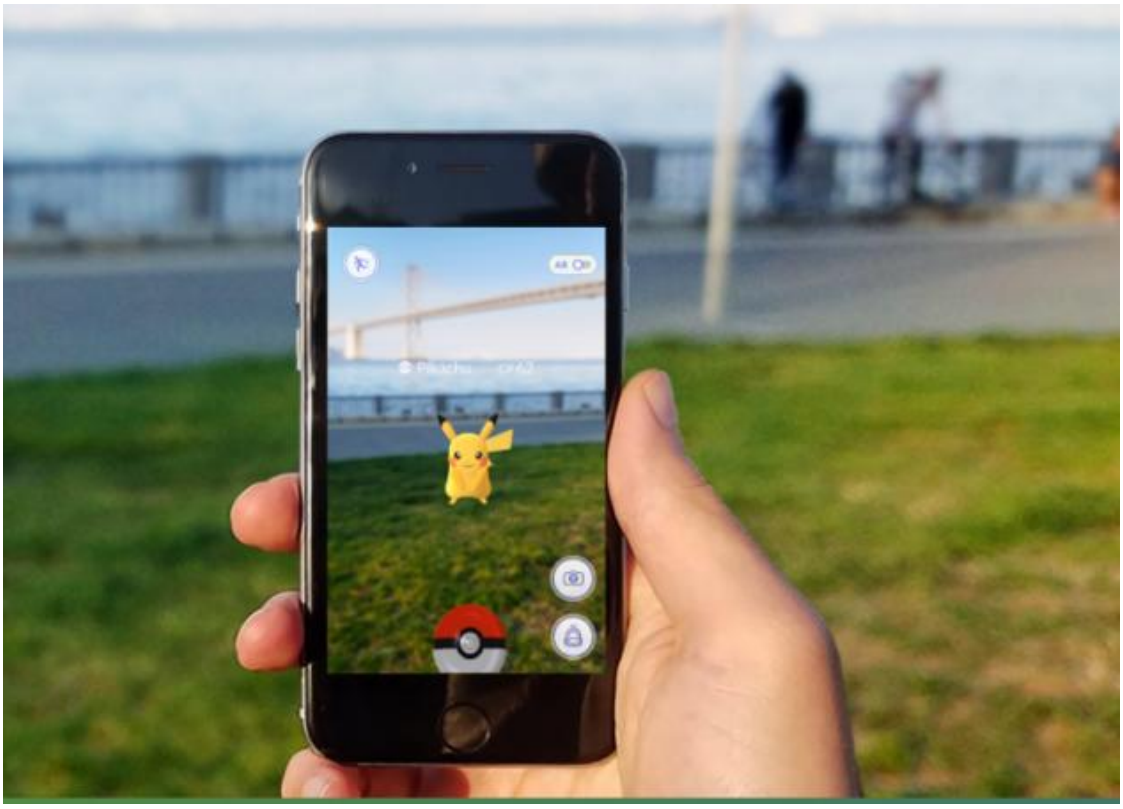

Рисунок 1.2 – Ігровий процес Pokemon Go

Тоді світ буквально охопила покемономанія – феномен, коли весь вільний час люди присвячували пошуку покемонів у будь-яких місцях – вдома, на вулиці, у школі, біля історичних пам’яток, та навіть у церквах. Це була не перша подібна гра, проте розробники Niantic, Inc. використали елемент ностальгії за серією коміксів та мультфільмів про покемонів, що були популярні близько 10 років тому, і це принесло великий комерційний успіх [2].

З того часу пройшло вже 3 роки, проте таких гучних “вибухів” ми більше не спостегірали, та популярність Pokemon Go також помітно спала.

Наразі популярні ігрові застосунки з використанням геолокації можна перелічити на пальцях.

1.2 Змістовний опис і аналіз предметної області

Щоб створити гру, треба визначитись з її жанром. Що буде метою гравця, його мотивацією? Що буде основною частиною геймплею? Жанрів існує безліч, і багато комп'ютерних ігор знаходяться на їх перетині. На рисунку 1.3 зображена відносно повна жанрова класифікація [3].

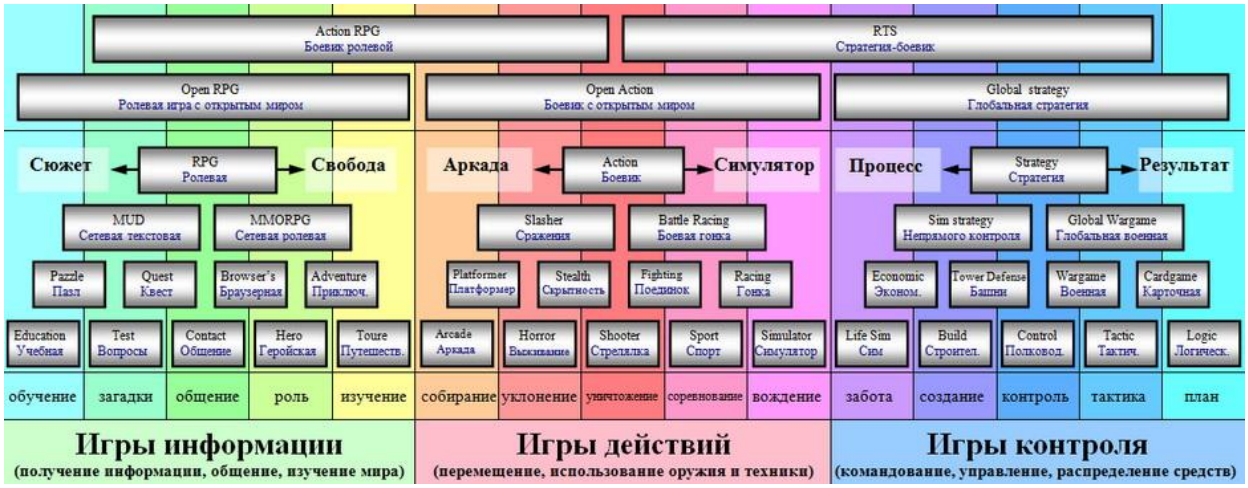

Рисунок 1.3 – Класифікація ігор за жанрами

Гра, створена в результаті виконання цієї роботи, також знаходиться на перетині декількох жанрів: екшн та стратегія в реальному часі.

Екшн (action в перекладі з англ. – дія) – жанр комп'ютерних ігор, де основним мотивом є експлуатація фізичних можливостей гравця: координації, швидкості реакції.

В екшн-іграх гравець зазвичай керує протагоністом (аватаром). Цей персонаж має знайти вихід з рівня, збирати предмети, уникати перешкод та змагатись із ворогами. Перешкоди та атаки ворогів впливають на здоров'я та кількість життів аватара. Коли життя закінчується, закінчується і гра для гравця, він отримує поразку. Інакше, якщо певна серія рівнів успішно пройдена, гравець перемагає. Іноді кількість рівнів може бути необмеженою, тож перемоги як такої немає. Тоді метою є набрати якомога більше очок.

До екшн-ігор можна віднести будь-яку гру, де перемога забезпечується фізичною перевагою. В таких іграх можуть бути присутні й інші елементи – гонки, головоломки, та інше, проте зазвичай вони прості, адже час є обмеженим.

Стратегія – для ігор цього жанру характерно те, що задля досягнення цілі необхідно використовувати стратегічне, тактичне, логічне мислення. мислення,

Стратегії класифікують за принципом течії часу. Існують стратегії в реальному часі (RTS, від англ. Real Time Strategy), покрокові стратегії (TBS, від англ. Turn Based Strategy) та гібридні варіанти. Важливо розуміти, що класифікація за принципом течії часу стосується лише часової організації в грі та не має стосунку до присутності чи відсутності таких елементів ігрового процесу, як будівництво бази, здобування ресурсів та подібних.

В стратегіях в реальному часі гравці діють одночасно, в покрокових – по черзі.

Окрім жанру, ігри також класифікуються за кількістю гравців. Вони бувають однокористувацькими та багатокористувацькими. За необхідністю інтернет-з’єднання – онлайн- та оффлайн-ігри.

Більшість ігор, що використовують геолокацію, є багатокористувацькими-онлайн стратегіями або екшн-іграми. Деякі з них використовують єлементи симуляції, рольових, пригодницьких ігор та інше.

Дана гра також є багатокористувацькою та потребує інтернет-з’єднання. Приналежність до жанру екшн визначають такі елементи, як наявність здоров’я та життів, рівнів (раундів) та змагання з іншими гравцями. Також певну роль грають фізичні показники – швидкість реакції та пересування. Стратегічним елементом є необхідність кооперуватися з іншими гравцями задля побудови найбільш успішної стратегії подолання

ворога. Оскільки всі гравці можуть діяти одночасно, це стратегія в реальному часі.

1.3 Аналіз успішних ІТ-проектів

1.3.1 Аналіз відомих технічних рішень

Оскільки ключовим елементом гри є геолокація, а важливою складовою ігрового процесу – постійний обмін координатами між гравцями та відображення їх на мапі, потрібно впевнитись, що це технічно можливо.

На щастя, нам не доведеться власноруч створювати програмні рішення і з нуля конструювати карту чи встановлювати зв’язок із супутником, а також перейматися юридичними аспектами приватності користувача. Як і для більшості сфер, для геолокації вже існують надійні обгортки від різних корпорацій. Оскільки даний проект реалізовується під iOS, нам цікаво розглянути рішення, яке пропонує Apple. Для роботи з локацією Apple створила нативний фреймворк CoreLocation.

CoreLocation надає сервіси для визначення географічного місцязнаходження пристрою. Для збирання даних фреймворк використовує все доступне апаратне забезпечення девайсу, у тому числі Wi-Fi, GPS, Bluetooth, магнітометр, барометр та забезпечення сотового зв’язку.

Коли додаток вперше надає запит на авторизацію, статус авторизації є невизначеним, і система просить користувача надати або заборонити запит на поширення місцязнаходження (рисунок 1.4). Система запам’ятовує вибір користувача та більше не відображає цю панель до того часу, доки запит не надійде знову.

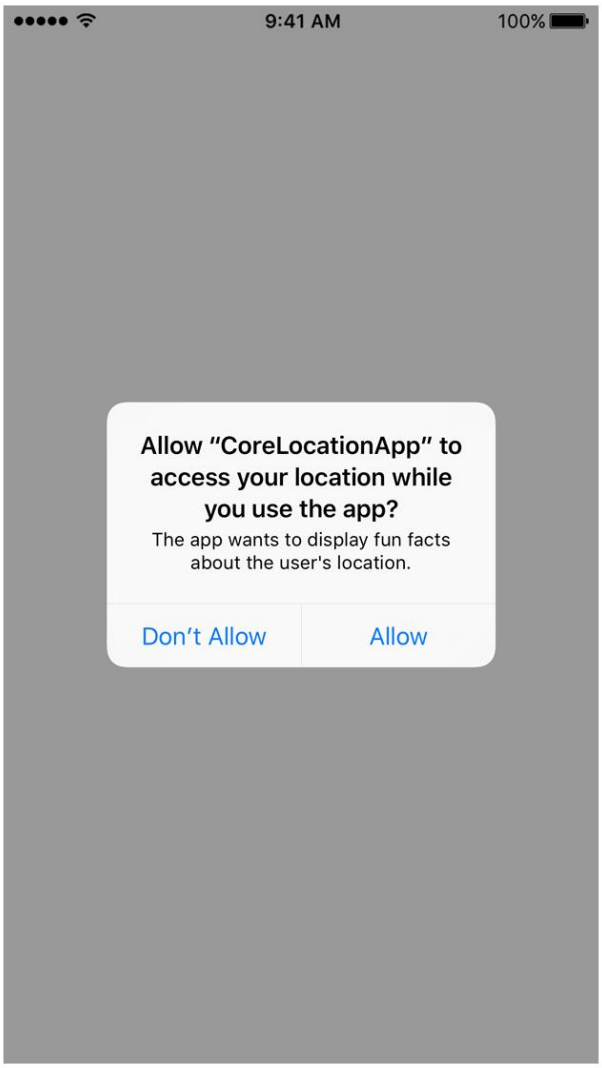

Рисунок 1.4 – Система запитує дозвіл

Для того, чтоб почати та припинити надавати застосунку дані, що стосуються локації, використовується об'єкт CLLocationManager.

Він використовується для налагодження, запуску та знищення сервісів CoreLocation. Об'єкт підтримує такі можливості:

– відслідковування значних або незначних змін в поточній локації користувача з певним ступенем точності, що можна налагодити;

– звітування про зміни напрямку з компасу девайса;

– моніторинг різних регіонів та генерацію івентів, коли юзер потрапляє

в них або залишає їх;

– заморожування надання івентів, коли додаток не використовується активно.

Щоб розпочати користуватись сервісами локації, потрібно виконати наступні кроки:

а) перевірити, чи додаток авторизований для того, щоб користуватись цими сервісами, і якщо ні, запросити дозвіл;

б) перевірити, які сервіси доступні для використання на конкретному девайсі;

в) створити об’єкт класу CLLocationManager та зберегти сильне посилання на нього; важливо зберігати це посилання до того моменту, доки не завершені всі завдання об’єкта;

г) присвоїти об’єкту делегат, що відповідає протоколу CLLocationManagerDelegate;

д) налаштувати властивості, що стосуються сервісів, що будуть використовуватись;

е) викликати потрібний метод щоб розпочати надавання подій.

Будь-які властивості потрібно наштовувати точно та обережно, адже CoreLocation, зважаючи на ці параметри, буде визначати, які апаратні частини повинні бути задіяні для їх задовільнення та оптимального використання заряду. Наприклад, якщо вистановити точність подій локації в 1км, то пристрій не буде користуватись GPS, адже сотових даних та Wi-Fi буде цілком достатньо. Це може суттєво заощадити енергію.

Усі оновлення стосовно локації та напрямку надаються делегату [4].

Для роботи з картами Apple надає розробникам нативний фреймворк MapKit. Із його допомогою можна відображати в додадтку мапи або супутникові зображеня та взаємодіяти с ними.

MapKit використовують щоб напряму вбудувати карти у вікна програми. На них додавати анотації та додаткові шари для позначок місця або напрямку [5].

Цими фреймворками користуються такі відомі додатки як Telegram та Uber, що свідчить про їх надійність. Їх функціоналу цілком достатньо, щоб задовольнити вимоги до даного проекту.

1.3.2 Аналіз відомих програмних продуктів

Серед найвідоміших продуктів із цієї області слід відзначити **Pokemon Go** (рисунок 1.5), про яку вже було згадано вище – безкоштовну багатокористувацьку рольову мобільну гру на основі визначення місцязнаходження доповненої реальності, створену компанією Niatic для iOS та Android. У грі гравці використовують мобільні пристрої з GPS аби знаходити, змагатися та тренувати віртуальних істот, яких називають покемонами, що з’являються на екрані так, ніби вони знаходяться на тому ж реальному місці, що і гравець. Гра підтримує внутрішні покупки додаткових предметів.

Рисунок 1.5 – Лого гри

Лише за тиждень після релізу гра здобула велику популярність. Відгуки від критиків були різні. Серед позитивних: концепція гри, заохочення до фізичної активності, популяризація ігор на основі локації та доповненої реальності. До негативних належали зауваження щодо технічних помилок (що були дуже помітні в перших версіях), а також те, що захоплення

грою спричило аварії та подейкуди порушення суспільного порядку. Так, Іран повністю заборонив гру на своїй території, а в Росії кілька людей потрапили до в’язниці за те, що ловили покемонів у церкві.

Попри суперечливі рецензії, гра швидко стала глобальним явищем та одним з найбільш використовуваних мобільних додатків. Її завантажили більш ніж 100 мільйонів людей з усього світу, а загальний прибуток склад понад 440 мільйонів долларів (при тому, що покупки робили лише близько 5% користувачів).

Pokemon Go мала кілька факторів успіху. Головний з них полягає в концепції. Покемони – милі казкові створіння, що нагадують нам дитинство (рисунок 1.6). 10 років тому максимальним рівнем поринання в реальність покемонів був перегляд мульфільмів та гра на приставці. Тоді ми й уявити не могли, що зможемо взаємодіяти з ними так близько, та майже мати можливість доторкнутись до них – адже ось вони, блукають зі мною вулицями мого міста!

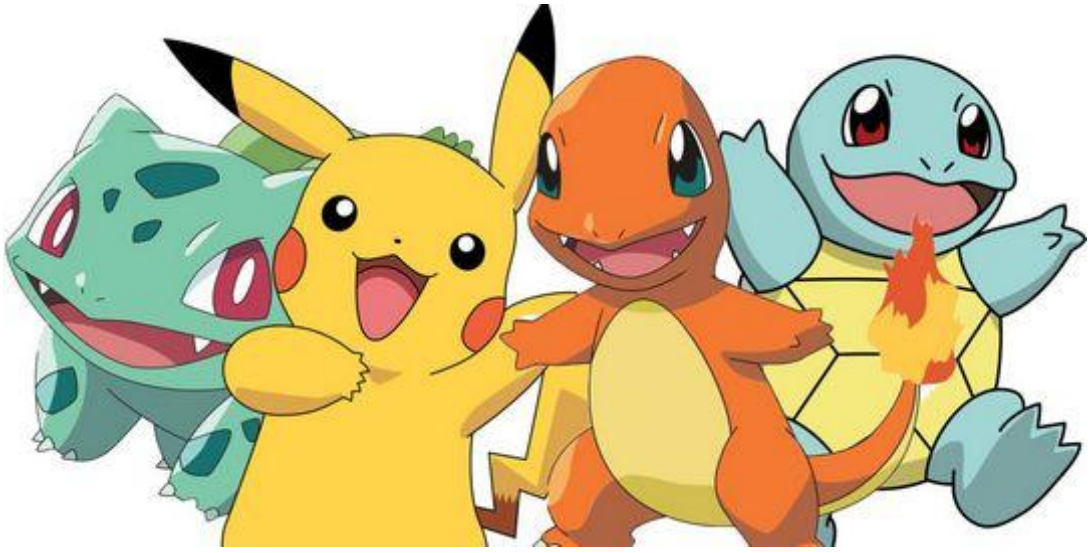

Рисунок 1.6 – Деякі покемони

Так, ностальгічний фактор мав великий вплив на популярність продукту. Серед інших факторів – великі капіталовкладення. Компанія Niatic народилась як стартап у межах Google, тож їм легко було залучити мільйонні

інвестиції від “батька”. Це якісно вплинуло на швидкість розробки та рекламну кампанію.

Важливо зазначити, що Pokemon Go належить до цілої серії офіційних ігор про покемонів. Ця серія за довгі роки існування сформувала навколо себе чисельну базу лояльних користувачів, які залюбки пробують нові ігри та розповідають про це в соціальних мережах. Ефект сарафанного радіо зіграв свою роль.

Здається, що популярності Pokemon Go завдячує саме доповненній реальності, проте Niatic вже створювала ігри з використанням AR, які й близько не мали такого успіху. Саме колаборація з Pokemon стала вирішальною.

Хоч яким би прекрасним не було почуття ностальгії, проте воно приходить та йде, і втримати користувачів 21-го сторіччя, спокушених гіпер-реалістичною графікою та неймовірно-закрученими сюжетами, лише цим почуттям не вдасться. Такими бонусами Pokemon Go похизуватись не може, тож до 2019 року хвиля її популярності не дісталася.

Також мінус Pokemon Go був у тому, що основна взаємодія гравця відбувалась із віртуальними істотами. Це швидко набридає. Взаємодіяти зі справжніми людьми, або навіть зі своїми друзями – значно цікавіше [2].

Інший продукт розробника Niatic більш схожий за задумом на той, що був створений в результаті виконання цієї роботи. Мова йде про гру **Ingress.** На ринку вона з’явилась у 2012 році. У 2015 році кількість гравців налічувала 7 мільйонів.

На тлі Pokemon Go це здається провалом, проте відносно інших ігор на ринку її можна назвати успішною. Розглянемо та проаналізуємо основні моменти геймплею.

Пересуваючись зі своїми мобільними пристроями в реальному світі, гравці повинні захоплювати у віртуальному світі гри *портали*, на місці яких

в реальному світі знаходяться зазвичай визначні місця. Гравці поділені на дві фракції, для яких геймплей не відрізняється – відмінності проявляються тільки через сюжетну лінію.

Важливим елементом ігрового процесу є Екзотична Матерія – певна енергія, якою володіють гравці та портали. Для захоплення порталів та більшості інших дій використовується саме вона. В процесі захоплення порталів гравці підвищують свій рівень.

Гравець представлений на ігровій мапі стрілкою з колом з радіусом 40 метрів. Мапа також містить обриси будинків та доріг, точки матерії, портали та інші ігрові елементи. Елементи мають колір своєї фракції.

Гра також має заплутаний науково-фантастичний сюжет із різними конспірологічними теоріями.

Ingress мотивує користувачів подорожувати, гуляти, відвідувати нові місця, зокрема пам’ятники, станції метро, дитячі та спортивні майданчики, фонтани, клумби, граффіті, різноманітні будівлі – адже портали знаходяться саме там. При зростанні рівня гравця йому доводиться відвідувати все більше нових локацій, адже рівень зростає при зростанні кількості відвідуваних та захоплених унікальних порталів та пройдених місій. Окрім того, на високих рівнях гра підштовхує до кооперації між гравцями, стимулюючи спільні заходи та прогулянки.

Інтерфейс гри зображено на рисунку 1.7.

Досвідчені гравці мають при собі зовнішні аккумулятори для мобільних пристроїв, адже при активній грі середньостатистична батарея телефона витримує не більше пари годин.

Модель монетизації побудована на внутрішньоігровій валюті, яка дозволяє придбати бонси для спрощення виконання місій та іншого [6].

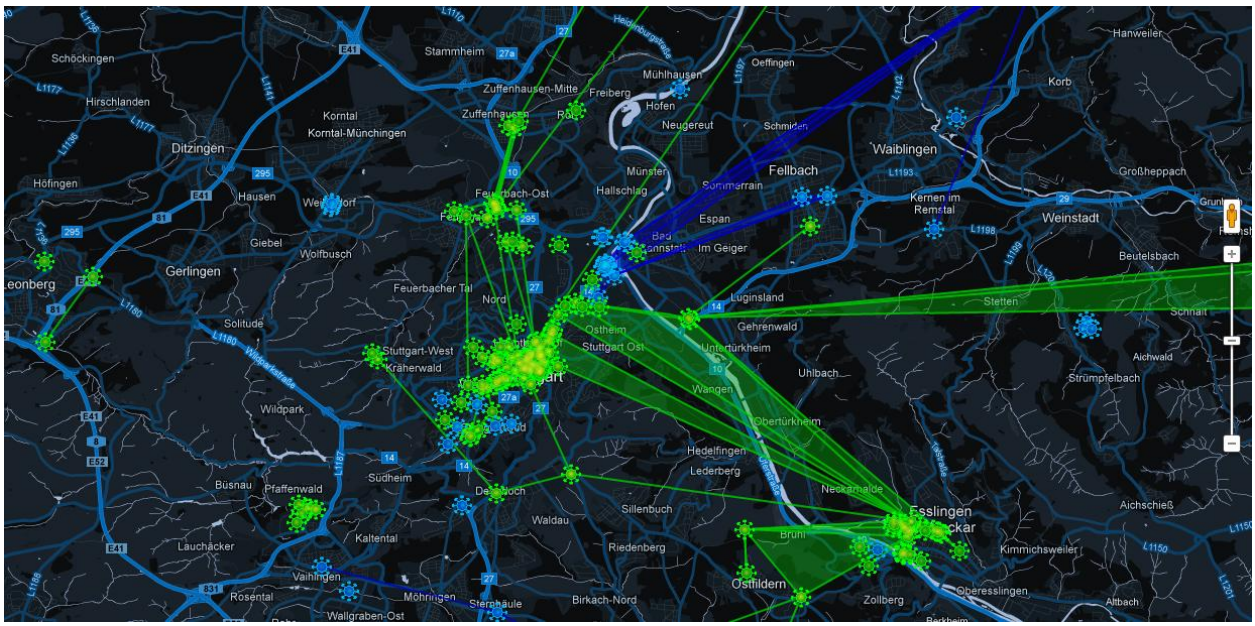

Рисунок 1.7 – Карта порталів в Ingress

Цікаво, що Ingress стала свого роду прототипом для Pokemon Go. На місці потралів Ingress, в Pokemon Go знаходяться так звані “покестопи”. Здавалося б, нащадок виявився успішнішим за пращура, проте порівнювати їх лише за кількістю користувачів було б несправедливо.

Ingress та Pokemon Go мають спільний недолік – відносно високий поріг вхождення. З цієї перспективи, закручений сюжет Ingress є недоліком, адже не можна просто взяти та почати грати – треба розібратись, які у кожної команди цілі, до кого приєднатись, як допомогти досягнути тих цілей. Тобто вже з самого початку користувач повинен прикладати певні зусилля.

Щодо Pokemon Go – спіймати першого покемона не так вже й важко, а от для усіх подальших доведеться розбиратись в тонкощах гри, адже кожний покемон має свої характеристики, і для кожного потрібен свій набір інструментів. Різноманітних інструментів та покемонів у грі ну дуже багато, і гравцю починає здаватись, що він ніколи не зможе розібратись в усьому. Це створює неприємний тиск.

Надлишкову кількість усіляких предментів, що лише ускладнюють вхождення в гру, має й Ingress.

Наведемо порівняльну характеристику ігор в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика продуктів

	Ingress	Pokemon Go
Сюжет	є	немає
AR	немає	є
Поріг вхождення	високий	високий
Цільова аудиторія	десятки мільйонів	сотні мільйонів
Взаємодія з іншими гравцями	є	немає

Використовуючи досвід попередників, можна зробити ставку на вдосконалення моментів, що були недостатньо продумані в них.

1.4 Аналіз вимог до програмного забезпечення

Гра має тільки один тип користувачів – власне, гравця. За концепцією, усі гравці мають рівні можливості, тобто жодного розподілення на ролі немає, Відмінності між гравцями можуть полягати лише в команді, до якої вони належать, та в рівні “життєвої сили”. Оскільки участь у грі означає, що людина ділиться своєю локацією з іншими людьми, задля безпеки повинна бути можливість припинити гру в будь який момент.

Геймплей має бути інтуїтивно зрозумілим, позбавленим будь-яких зайвих деталей. Адже ідея гри полягає саме в її простоті. Жодного сюжету, предметів, героїв і так далі – просто бери та грай.

1.4.1 Розроблення функціональних вимог

Гра повинна мати такий функціонал:

- вхід через Facebook (таблиця 1.2);

- встати в чергу на раунд (таблиця 1.3);
- відображення гравців на карті (зі збереженням анонімності);
- пауза (обмежена к-сть разів на день) (таблиця 1.4);
- атака слабшого гравця при попаданні в його зону (таблиця 1.5);
- кооперація в атаці (таблиця 1.5).

Продукт представляє собою клієнт-серверний застосунок. Розглянемо варіанти використання клієнтської частини задля деталізації вимог.

Таблиця 1.2 – Варіант використання GE001

Назва	Вхід через Facebook
Опис	Користувач має можливість авторизуватись в системі, використовуючи свою сторінку Facebook.
Передумови	Неавторизований користувач відкрив додаток.
Постумови	Користувач авторизований.
Основний сценарій	а) користувач натискає на кнопку “Вхід через Facebook”; б) додаток відкриває браузер або додаток Facebook (якщо встановлено); в) Facebook питає в користувача дозвіл на використання його імені та e-mail; г) якщо користувач погоджується, авторизує його (якщо це відбулось вперше – додає до бази даних); д) користувача повертає в додаток, де він вже авторизований.

Таблиця 1.3 – Варіант використання GE002

Назва	Почати гру
Опис	Щоб розпочати гру, гравець встає в чергу на формування команд. Коли команди сформовано, гра автоматично розпочинається.
Передумови	Користувач авторизувався.

Постумови	Гра розпочалась.
Основний сценарій	а) гравець натискає кнопку “Встати в чергу”; б) гравця додано в чергу на формування команд;

Продовження таблиці 1.3

Основний сценарій	в) коли в черзі набрано 20 людей, гравця випадково розподіляє в одну з 2 команд по 10 людей; г) гра автоматично розпочинається.

Таблиця 1.4 – Варіант використання GE003

Назва	Поставити на паузу
Опис	Гравець в будь-який момент має можливість поставити гру на паузу (припинити надавати свою локацію та бачити локацію інших).
Передумови	Користувач розпочав гру.
Постумови	Для користувача гра поставлена на паузу або завершена.
Основний сценарій	а) натиснути кнопку пауза; б) система перевіряє: якщо ліміт пауз на день ще не вичерпано, гра ставиться на паузу; якщо вичерпано – гра для цього гравця закінчується.

Таблиця 1.5 – Варіант використання GE004

Назва	Атака
Опис	Гравець попадає до зони гравця з меншим рівнем життєвої сили (aka – жертви), і розпочинає атаку
Передумови	Користувач наблизився до жертви
Постумови	Жертву “вбито”, користувачу налічується сила
Основний сценарій	а) увійти в зону атаки жертви (50 метрів); б) знаходитись у зоні атаки доки жертва повністю не

	залишилась без сили (5 хвилин); в) якщо поряд є гравці тієї ж команди, відбувається кооперація; в такому випадку, час розділиться на кількість скооперованих гравців. г) жертва “вбита” – для неї гра завершується; д) учасникам атаки нараховується до 12 одиниць сили.

Об’єднаємо сценарії використання у діаграмі прецедентів (рисунок 1.8).

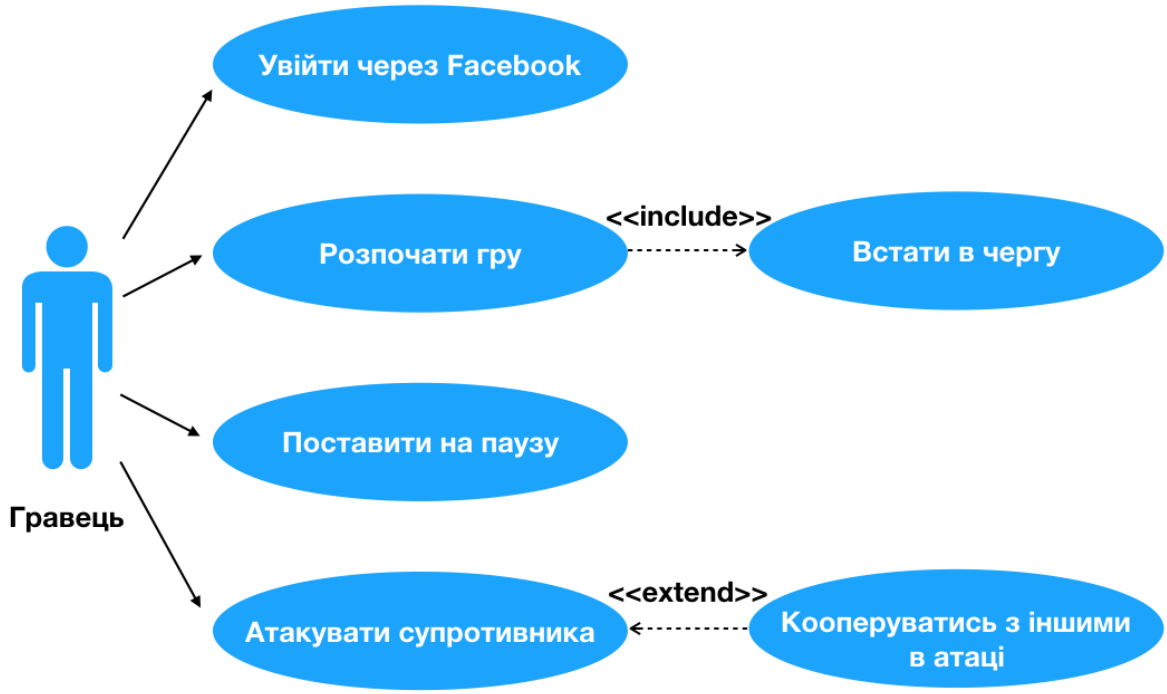

Рисунок 1.8 – Use-case діаграма

Використаємо матрицю трасування (таблиця 1.6) для наочності відповідності між вимогами та варіантами використання.

Таблиця 1.6 – Матриця трасування

	Увійти через Facebook	Розпочати гру	Поставити на паузу	Атакувати супротивника	Встати в чергу	Кооперуватись з іншими в атаці
GE001	■					
GE001		■			■	
GE002			■			
GE004				■		■

Обчислювальні задачі та задачі з обміном геолокацією між гравцями, а також зберігання бази користувачив виконуються на сервері.

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення повинне відповідати наступним нефункціональним вимогам:

– локалізація інтерфейсу – можливість відображати всі тексти будь-якою мовою;

– підтримувана версія iOS – 11.0 і вище;

– передача даних через захищені канали;

– анонімність користувачів.

1.4.3 Постановка комплексу завдань модулю

Розроблена гра має нести розважальний характер. Головна її мета – отримання насолоди від ігрового процесу. Серед найважливіших завдань розробки – спонукання до фізічної актиності (пересування в реальному світі) та коммунікації (для кооперацій в грі).

Щоб гравці отримували задоволення, нашим завданням є забезпечити їх безпеку та комфорт.

Заходи безпеки:

- можна припинити надавати свою локацію в будь-який момент;
- можна відзвітувати небезпечну поведінку гравця;
- гра не розголошує спражнє ім’я, фото чи будь-які контакти гравців;
- усі приватні дані передаються через захищені з’єднання та зберігаються у зашифрованому вигляді;

Комфорт забезпечується швидким обміном даними, інтуїтивним UX та відсутністю зайвих деталей.

Мета розробки – підвищити користь від мобільних ігор, задіявши в них реальний світ, та зробити це краще, ніж це робили конкуренти (доступніше та цікавіше).

Для цього потрібно виконати наступні задачі:

- розробити UX клієнта;
- розробити клієнт-серверну архітектуру;
- створити клієнт на iOS;
- створити сервер.

1.5 Висновки до розділу

В цьому розділі була проаналізована предмента область, технології, що можуть бути використані, а також конкуренти. На основі цього аналізу був розроблений ряд вимог до даного проекту.

Перша вимога: проект повинен бути розважальним. Відчуття задоволення, азарту, розвитку повинне супроводжувати гравця. Для того, щоб цей ефект зберігався, на даний момент гру вирішено не доповнювати сюжетами чи додатковими елементами. Поріг вхождення має бути низьким.

Друга вимога: гра має бути безпечною. Оскільки ми маємо справу з такими приватними даними, як локація людей, треба не допускати, щоб хтось міг дізнатися точне місцезнахождення гравців без їх дозволу.

Третя і остання вимога: гра має бути мотивуючою. Більшість із нас проводить мінімум 8 годин на день сидячи в офісі. Цей проект може мотивавути людей частіше виходити на вулицю та ходити пішки.

За цими вимогами був реалізований клієнт-серверний застосунок на iOS.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Вхідними даними для клієнта є власне взаємодія користувача з клієнтом. Користувач може надавати клієнту такі дані:

- інформацію з Facebook-сторінки (ім’я та e-mail);
- запит на вставання в чергу на гру;
- дозвіл/заборона на використання геолокації;
- запит на припинення використання локації під час гри;
- запит на вихід з гри.

Вхідними даними до серверної частини застосунку є запити від клієнта на:

- авторизацію за даними Facebook;
- вставання в чергу на гру;
- оновлення локації;
- вихід з гри;
- атаку суперника;
- кооперацію в атаці;
- заборону на транслювання даних локації.

Для моделювання програмного забезпечення зобразимо схематично усі сценарії клієнт-серверної взаємодії. Для конструювання схем використаємо діаграми діяльності.

Нам знадобиться послідовний опис кожного сценарія.

Сценарій авторизації в додатку:

а) користувач натискає кнопку “Вхід через Facebook”; його перенаправляє на сторінку Facebook, де він погоджується на використання своїх даних; на сервер надсилаються його дані;

б) сервер перевіряє, чи є ці дані в базі; якщо так, користувача авторизовано, він переходить до роботи зі своїм аккаунтом у грі; якщо ні, його дані записуються в базу, створюється новий аккаунт; після цього користувача також авторизовано.

Діаграму діяльності цього сценарію зображено на рисунку 2.1.

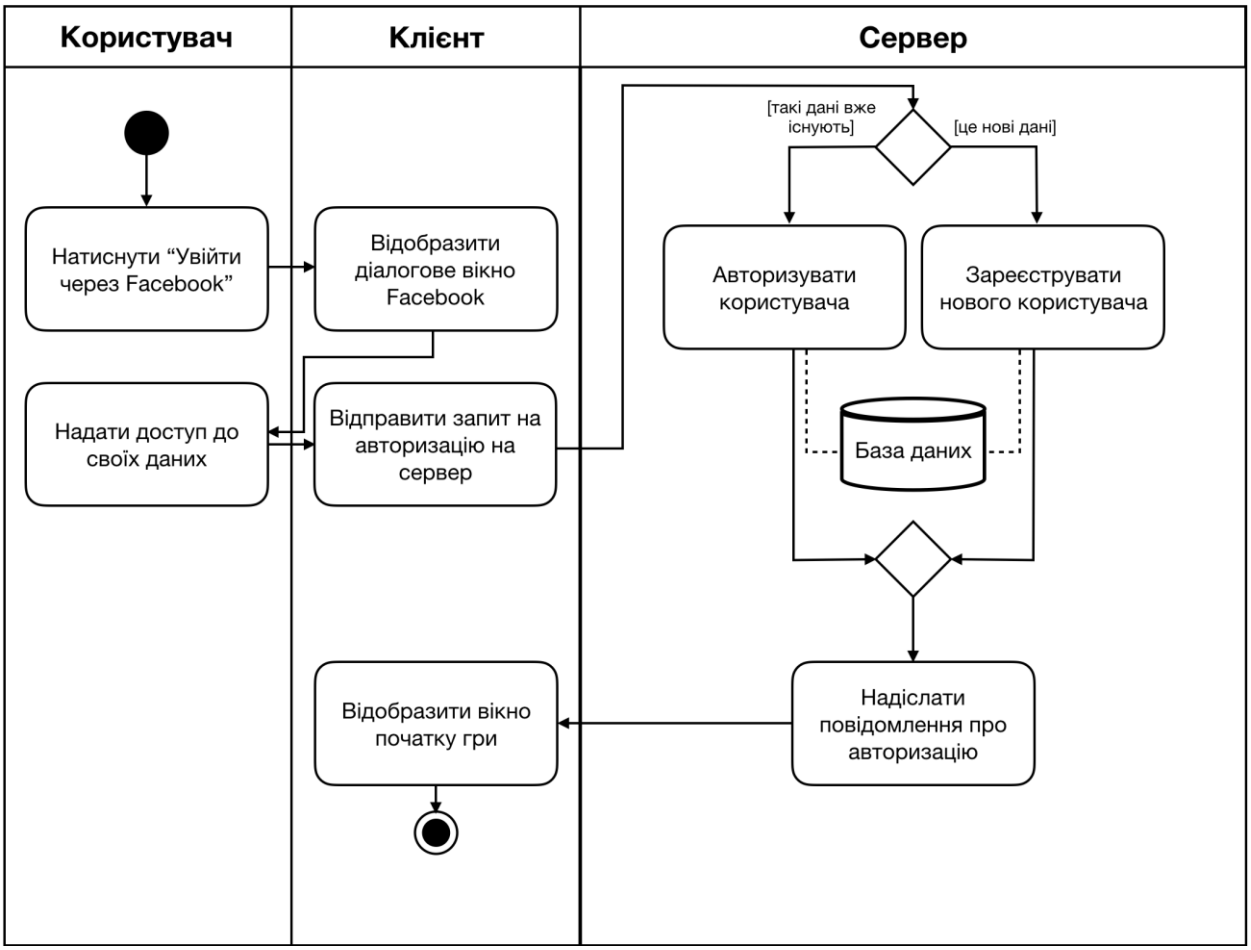

Рисунок 2.1 – Діаграма діяльності авторизації користувача

Сценарій початку гри:

а) користувач натискає кнопку "Встати в чергу";

б) на сервер відправляється відповідний запит;

в) користувач отримує відповідь, що його успішно поставлено в чергу, та поточну кількість людей в черзі;

г) коли у черзі з'являється нова людина, сервер відправляє клієнту нову

кількість і лічильник збільшується;

д) коли у черзі набралося 20 людей, сервер випадково розбиває їх на дві

команди, відправляє клієнту повідомлення про початок гри та колір команди.

Діаграму діяльності цього сценарію зображено на рисунку 2.2.

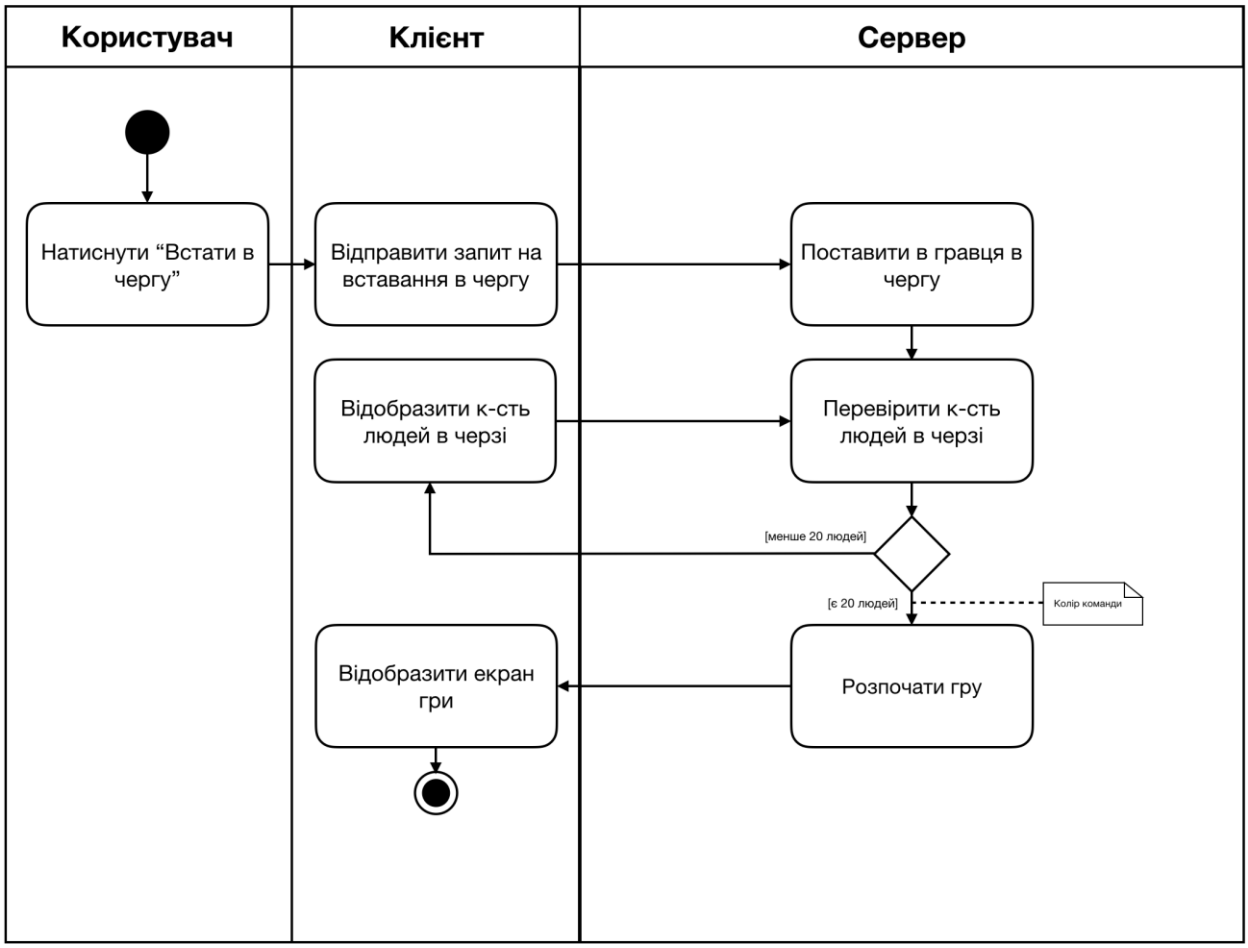

Рисунок 2.2 – Діаграма діяльності початку гри

Відразу при початку гри потрібно зібрати та відобразити локації користувачів. Після початку цей сценарій потрібно повторювати з певною регулярністю, щоб гравці мали свіжу інформацію про місцезнаходження всіх учасників гри.

Сценарій оновлення локації:

а) клієнт автоматично відправляє свою поточну локацію та запит на оновлення локацій інших;

б) сервер отримує нові дані та запит; нові дані записуються в базу;

в) із бази беруться поточні дані інших гравців та відправляються на клієнт;

г) клієнт отримує оновлені дані.

Діаграму діяльності цього сценарію зображено на рисунку 2.3.

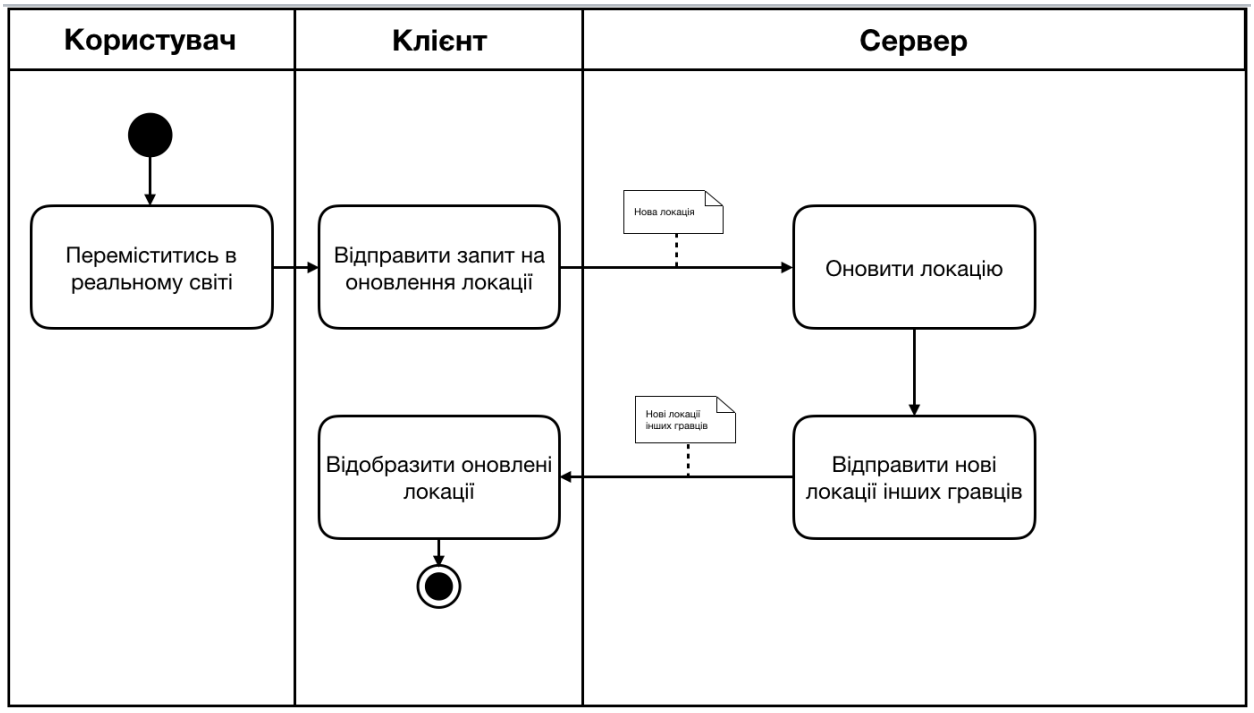

Рисунок 2.3 – Діаграма діяльності оновлення локації

Найбільш складним, та водночас найбільш важливим сценарієм є сценарій атаки. Він потребує швидкого оновлення даних, взаємодію сервера одразу з кількома клієнтами, часте звертання до бази даних, роботу з великою кількістю даних. Це все дає велике навантаження на систему, і повинно буде змодельовано з особливою уважністю, адже саме відмінне виконання цього сценарію забезпечить плавний та приємний ігровий процес.

Сценарій атаки:

а) клієнт відправив локацію;

б) сервер робить перевірку: якщо гравець знаходиться на відстані 50 або

менше метрів від гравця з іншої команди з меншим рівнем сили, сервер відправляє повідомлення про початок атаки;

в) якщо в зоні атаки з’явився ще один гравець з команди нападника, атака пришвидшується з певним коефіцієнтом;

г) якщо в зоні атаки з’явився ще один гравець з команди жертви, атака сповільнюється з певним коефіцієнтом;

д) кожний момент часу клієнти отримують повідомлення, скільки часу залишилось до завершення атаки, а також поточний рівень життєвої сили жертв, що падає;

е) з кожним оновленням локації від обох учасників сервер перевіряє умову атаки. Якщо умова перестала виконуватись, атака зупиняється;

ж) якщо час вийшов, сервер відправляє повідомлення до клієнтів-нападників, що атака завершена успішно і їм нараховано певну к-сть балів життєвої сили; клієнти-жертви отримують повідомлення, що гра для них завершена; в базі даних оновлюється статус жертв, що “вбиті”, а також рівні життєвої сили всіх учасників атаки, що вижили.

Діаграму діяльності цього сценарію зображено на рисунку 2.4.

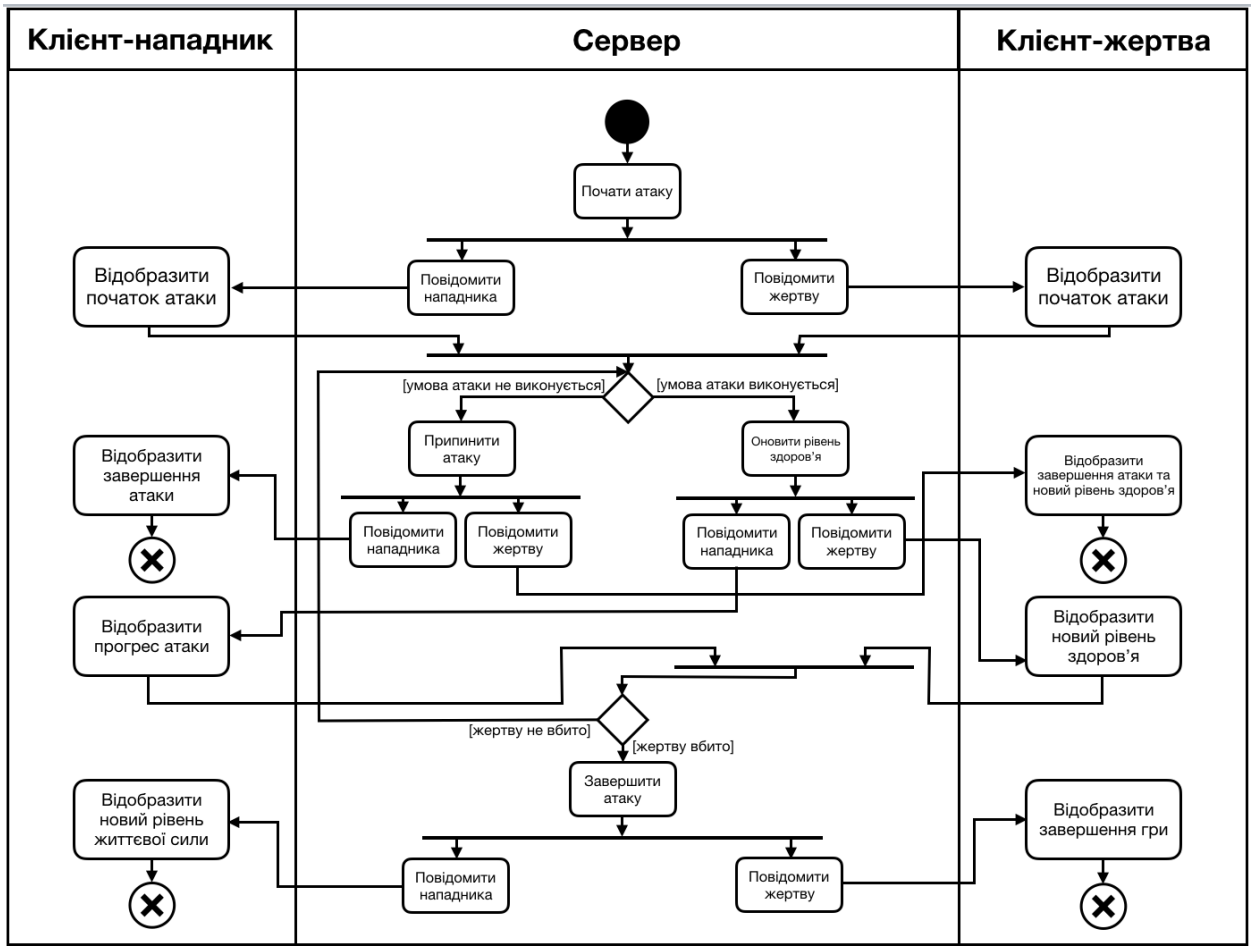

Рисунок 2.4 – Діаграма діяльності атаки

Щоб уникнути нагромадження елементів та нечитабельності, схему атаки зобразимо спрощено, без урахування можливості кооперації (участі в атаці більш ніж 2 гравців).

2.2 Архітектура програмного забезпечення

Для створення даного застосунку потрібно сконструювати клієнтську та серверну частини. Серверна частина складається з власне сервера з бізнес-логікою, бази даних та API для взаємодії з клієнтом. Клієнтський застосунок має містити графічний інтерфейс для взаємодії з користувачем, власне, бізнес логіку (мало, адже більшість обчислювань буде відбуватись на сервері, а клієнт майже у всіх сценаріях використовується лише для відображення даних), а також частина взаємодії з мережею через API, що є своєрідним

прошарком між сервером та клієнтом. Будь-яке звернення до мережі повинне відбуватись через неї.

Для побудови iOS-клієнта використовується фреймворк CocoaTouch мовою Swift. Він містить усі необхідні нативні UI-елементи. Сервер також написано мовою Swift з використанням фреймворка Perfect.

Дані передаватимуться через протокол HTTPS, що є розширенням протоколу HTTP з підтримкою криптографічного шифрування. Він використовує криптографічні протоколи SSL та TLS.

Для спрощення роботи з мережею використаємо опен-сорсний фреймворк Alamofire. Він робить запити набагато «читабельнішими», скорочує обсяг коду та бере на себе всю роботу з протоколами та токенами.

Для створення клієнтського додатку на iOS буде використовуватись архітектурний шаблон MVVM.

Великою перевагою MVVM перед MVC є маленькі класи та набагато більше розподілена відповідальність. Код MVVM є читабельним, а елементи шаблону – model, view та viewModel, є замінними. Тобто, при необхідності, можна лишити модель та представлення, проте змінити viewModel, і тоді логіка роботи систми зміниться. Те саме стосується інших елементів.

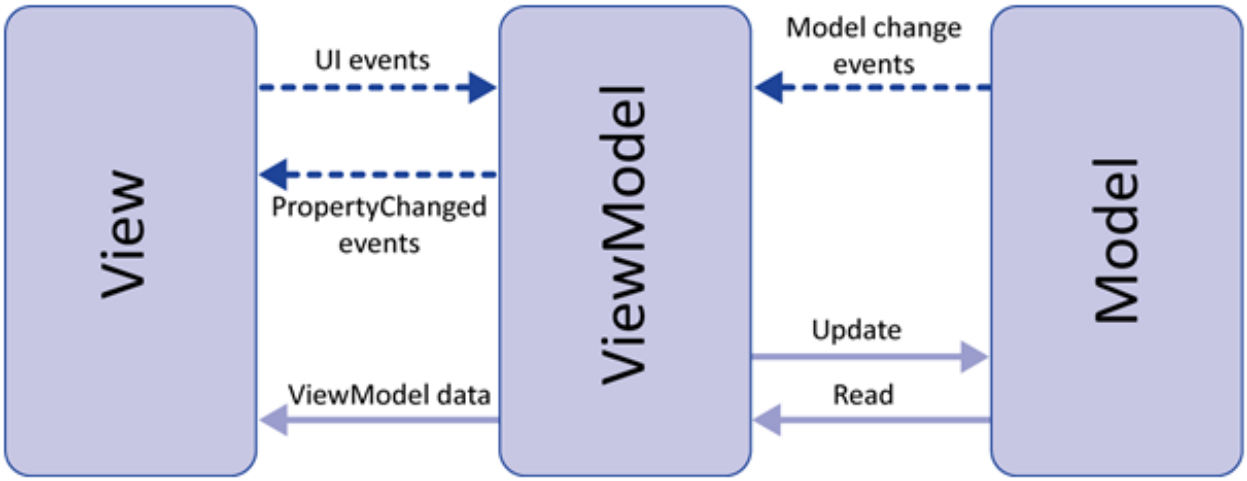

Рисунок 2.4 – Структура шаблону MVVM

Даний шаблон складається з трьох частин:

– модель (model) – містить логіку роботи з даними та опис

фундаментальних даних, що потрібні для роботи застосунку. Власне, майже вся бізнес-логіка має знаходитись в моделі;

– представлення (view) – графічний інтерфейс (вікна, кнопки, таблиці і так далі). Його єдина задача – відображати інформацію, що надає ViewModel. Він слухає події ViewModel, і, якщо відбувається будь-яка зміна, представлення реагує на неї відповідними змінами в інтерфейсі. І навпаки, якщо користувач взаємодіє з представленням, він повідомляє про це ViewModel та визиває відповідну команду. Представлення не може взаємодіяти з моделлю напряму;

– модель представлення (viewModel) – шар між view та model, що відповідає за їх взаємодію. Ця частина містить модель та є обгорткою даних з неї, а також містить команди, якими може користуватись представлення задля впливу на модель [8].

2.3 Конструювання програмного забезпечення

База даних

Розробимо базу даних, що міститеме необхідну інформацію для сервісу.

Уся перманентна інформація про юзера зберігається в одній таблиці (та, що не змінюється під час гри). Це його унікальний ідентифікатор, ім’я, електронна пошта, загальна кількість балів. Схему бази даних зображено на рисунку 2.5.

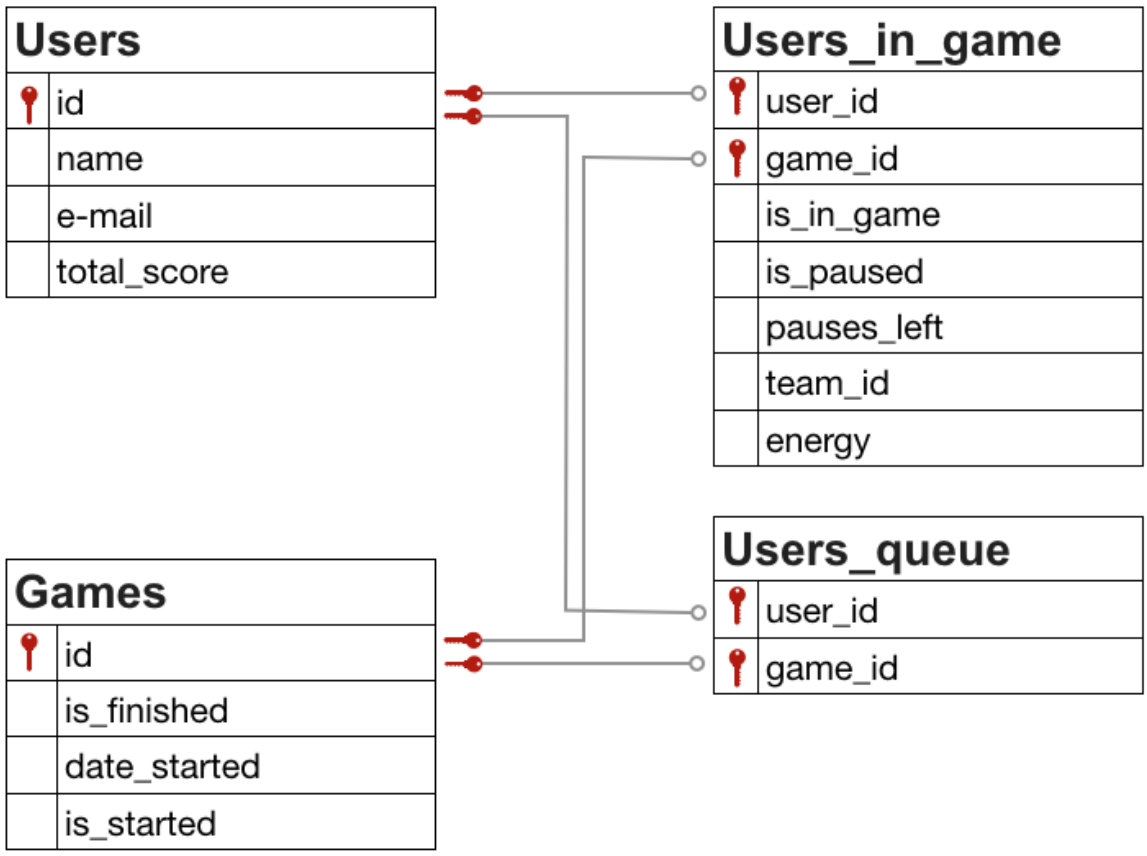

Рисунок 2.5 – Схема бази даних

Інформація, що стосується поточної гри, знаходиться в окремій таблиці. В ній зберігаються дані, до якої команди належить юзер, чи активно він зараз у грі, його рівень життєвої сили та інше.

Інформація про користувачів, що чекають на початок гри в черзі, знаходиться у таблиці Users_queue.

Класи та інтерфейси

Розглянемо основні класи клієнтського застосунку, їх наповнення та інтерфейси.

Фундаментальним типом застосунку є гравець – Player. Цей тип має містити всю необхідну інформацію про гравця для даної гри – його команду, локацію, його ідентифікатор, ім’я та рівень сили. Усі ці поля є публічними та немутабельними задля уникнення можливих небажаних змін. Щоб змінити

локацію гравця, система створюватиме нового гравця з новою локацією щоразу, коли надходить оновлена інформація з сервера.

Player не містить жодних методів, він є лише збіркою інформації (таблиця 2.1).

Таблиця 2.1 – Поля класу Player

Поле	Тип	Призначення
Id	Int	Унікальний ідентифікатор гравця.
Name	String	Нікнейм, що гравець обирає сам.
Team	Team	Команда, до якої гравець належить у даній грі.
Location	CLLocationCoordinate2D	Поточні координати.
Power	Double	Поточний рівень життєвої сили.

Даний застосунок має містити абстракцію, що буде відповідати за постійне надання актуальної інформації для її відображення графічним інтерфейсом, а також надіслання нової актуальної інформації на сервер. Назвемо цю абстракцію GameEngine.

За головний екран гри відповідає класс GameViewController. Він власне містить карту та відображає на ній інформацію про гравців. Він є делегатом класу MapView, адже до його зони відповідальності належить все відображення.

Для роботи з Facebook використовується офіційне Facebook SDK. Воно містить особливу кнопку FBSDKLoginButton, яка вже містить усю необхідну прив’язану логіку.

Класс LoginViewController містить цю кнопку, а також відповідні делегатські методи, щоб реагувати на її події.

Наведемо основні методи класів та інтерфейсів у таблиці 2.2.

Таблиця 2.2 – Методи основних класів та інтерфейсів

Клас/Интерфейс	Метод	Опис
GameEngine	synchronize(player: Player)	Надсилає нові дані про поточного гравця на сервер.
	onPlayersUpdate(with completion: @escaping ([Player]) -> Void)	Виконує певні команди у момент запиту на оновлення інформації про гравців, надаючи цих гравців.
GameViewController	updatePlayers()	Через GameEngine отримує актуальну інформацію про гравців та відображає її на карті.
	draw(players: [Player])	Власне створює для кожного гравця графічний об’єкт та накладає його на карту.
	locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])	Делегатський метод менеджера локацій, викликається щоразу, коли локація користувача оновлюється, і дозволяє класу виконувати певні команди як реакцію на цю

		подію (в даному випадку – відправка нової актуальної інформації про локацію користувача на сервер).

Продовження таблиці 2.2

GameViewController	mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer	Делегатський метод mapView, викликається щоразу, коли карта відмальовує свої елементи. Тут клас кастомізує ці елементи (розмальовує різними кольорами гравців, тощо).
LoginViewController	loginButton(_ loginButton: FBSDKLoginButton!, didCompleteWith result: FBSDKLoginManagerLoginResult!, error: Error!)	Метод викликається, коли пройшов процес логіну. Параметри дозволяють дізнатись, чи є він успішним, а якщо ні – то що саме зашкодило, а отже зреагувати відповідно до ситуації.
	loginButtonDidLogOut(_ loginButton: FBSDKLoginButton!)	Метод, що викликається, коли користувач вийшов з аккаунту через кнопку Facebook. Також дозволяє виконати команди з реакції інтерфейсу на цю подію.
BeginViewController	startGame()	Приймає подію натиску на

		кнопку «Почати гру».
	loginButtonDidLogOut(_ loginButton: FBSDKLoginButton!)	Метод, що викликається, коли користувач вийшов з аккаунту через кнопку Facebook.

Продовження таблиці 2.2

QueueViewController	cancel()	Приймає подіє натиску на кнопку «Скасувати», і виконує відповідні команди viewModel.
PauseViewController	loginButtonDidLogOut(_ loginButton: FBSDKLoginButton!)	Метод, що викликається, коли користувач вийшов з аккаунту через кнопку Facebook. Також дозволяє виконати команди з реакції інтерфейсу на цю подію.
	backToGame()	Повертає гравця з паузи до гри.
	exitGame()	Виводить гравця з гри.
GameViewModel	pause()	Ставить гру на паузу, відображає паузу на клієнті та відправляє відповідний запит на сервер.
	move(marker: PlayerView, to: CLLocation)	Переміщає конретного гравця на його нові координати.
	attack(player: Player)	Виконує команду атаки.
	updatePlayers()	Відправляє актуальну

		інформацію про гравця та отримує нову інформацію про всіх інших, виконує команди оновлення графічного представлення.

Продовження таблиці 2.2

PlayerView	move(to: CLLocation)	Переміщає графічне представлення гравця на його нові координати на мапі.
	pulsate()	Пульсує, відображуючи процес атаки.

Для наочності відношень між класами розроблюваного ПО, наведемо діаграму класів на рисунку 2.6. Як видно з діаграми, було вирішено використати паттерн «Координатор», що відповідає за представлення потрібних екранів. Це робить систему гнучкою та зменшує кількість залежностей.

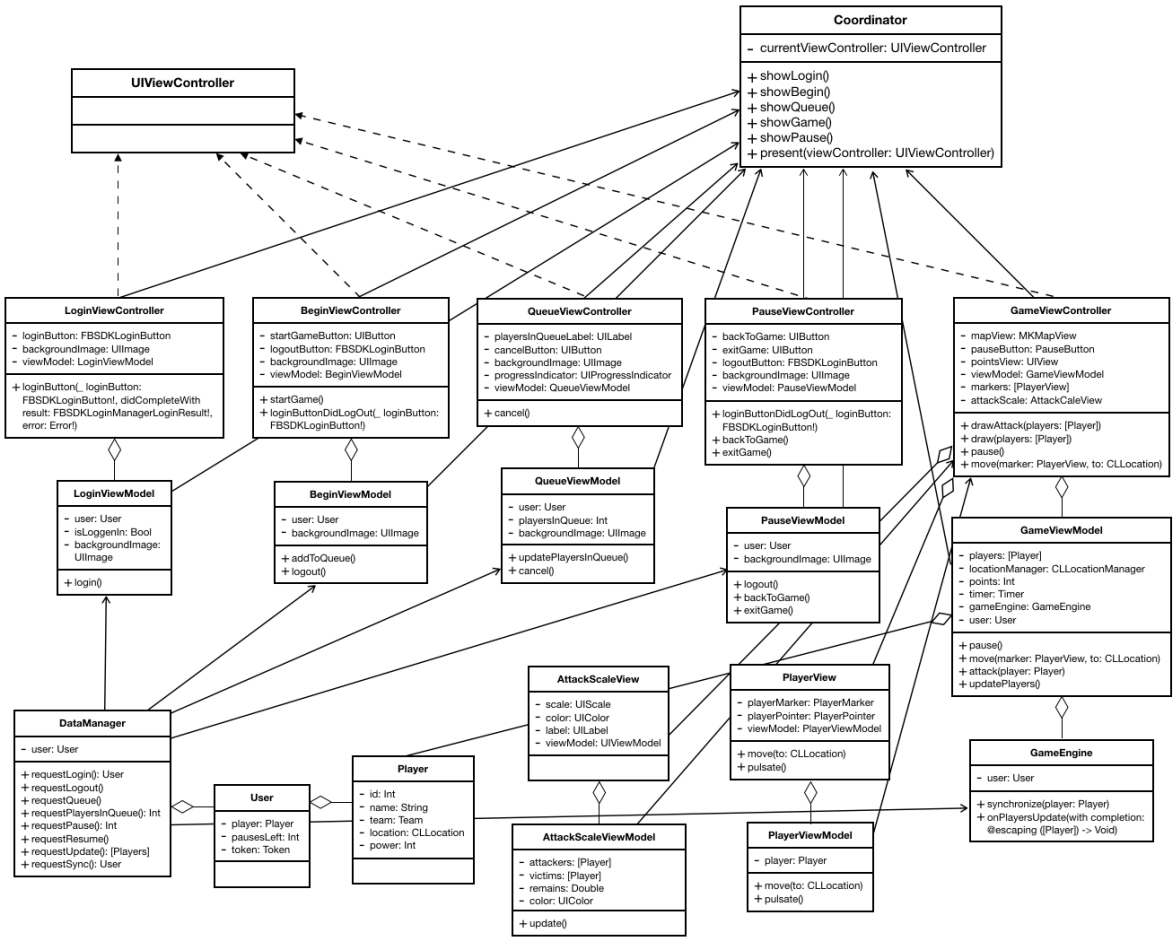

Рисунок 2.6 – Діаграма класів

Дизайн

Для гри, дизайн – велика частина приємного ігрового процесу. І мова не про те, що кнопки повинні бути приємного кольору, а про те, що будь-яка дія повина бути інтуїтивно зрозумілою. Добрий дизайн – це той, якого користувач не помічає. У розробки мобільних додатків є свої особливості дизайну – наприклад, найважливіщі кнопки слід розміщувати внизу екрана, щоб до них можна було дотягнутись пальцями.

Apple приділяє багато уваги дизайну, і цього ж вимагає від розробників на свої платформи. Вони мають спеціальну збірку HIG – human interface guidelines, що перекладається як «вказівки до людського інтерфейсу». Ця збірка містить вказівки до того, як повинні виглядати елементи графічного інтерфейсу, як повинні взаємодіяти з користувачем та реагувати на його дію.

Оскільки значна частка розробників дотримується цих вказівок, додатки на iOS зручні для користування, адже якщо є досвід хоча б з одним із них, інший вже буде зрозумілий за аналогіїю. Консистентність – ось що виділяє цю платформу серед інших [9].

При розробці дизайну було дотримано описаних вказівок.

Розглянемо макети наступних екранів програми:

- екран логіну (рисунок 2.7);
- екран початку гри (рисунок 2.8);
- екран черги (рисунок 2.9);
- екран паузи (рисунок 2.10);
- екран гри (рисунок 2.11).

Екрани логіну, початку гри, черги на паузи є за задумкою простими: вони містять декілька кнопок і все.

Натомість, екран гри має багато елементів. Маркери гравців мають колір команд, до яких належать ці гравці. На вказівниках до них вказано рівень їх сили. Кнопка «Пауза» знаходиться унизу, щоб до неї було зручно дотягуватись пальцем. Поруч із нею у кружечку відображена доступна кількість пауз.

Рисунок 2.7 – Екран логіну

Рисунок 2.8 – Екран початку гри

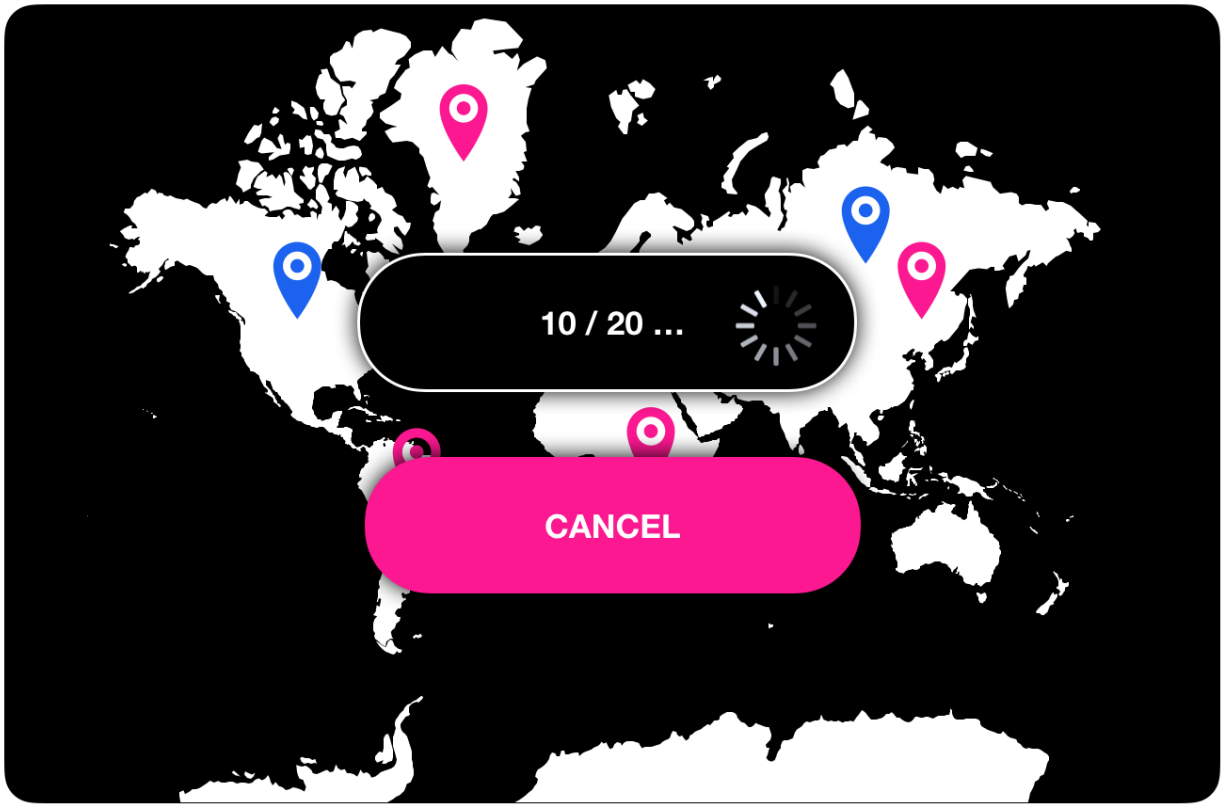

Рисунок 2.9 – Екран черги

Рисунок 2.10 – Екран паузи

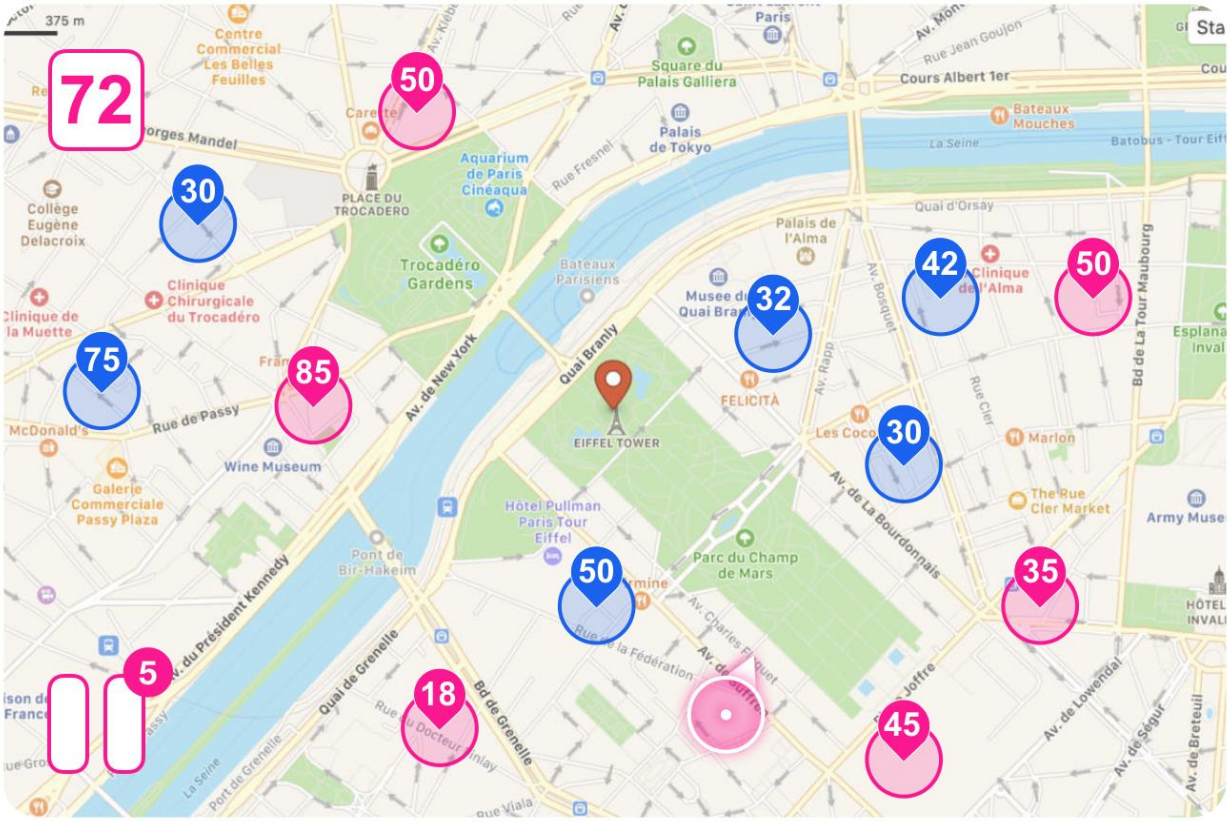

Рисунок 2.11 – Екран гри

Маркер власне гравця відображається білим кольором із підсвіткою кольору його команди. Також маркер гравця містить трикутник-компас, що відображає напрямок руху. Макер гравця (себе) не містить вказівника, свій рівень сили можна побачити в лівому верхньому куті.

Процес атаки відображається таким чином:

– маркери гравців, що беруть участь в атаці, починають рівномірно пульсувати;

– у верхньому правому куті з’являється шкала, де перелічені нападники

та жертви;

– рівень сили жертв зменшується;

– на шкалі відображається прогрес атаки.

Екрани процесу атаки та її завершення зображено відповідно на рисунку 2.12 та 2.13.

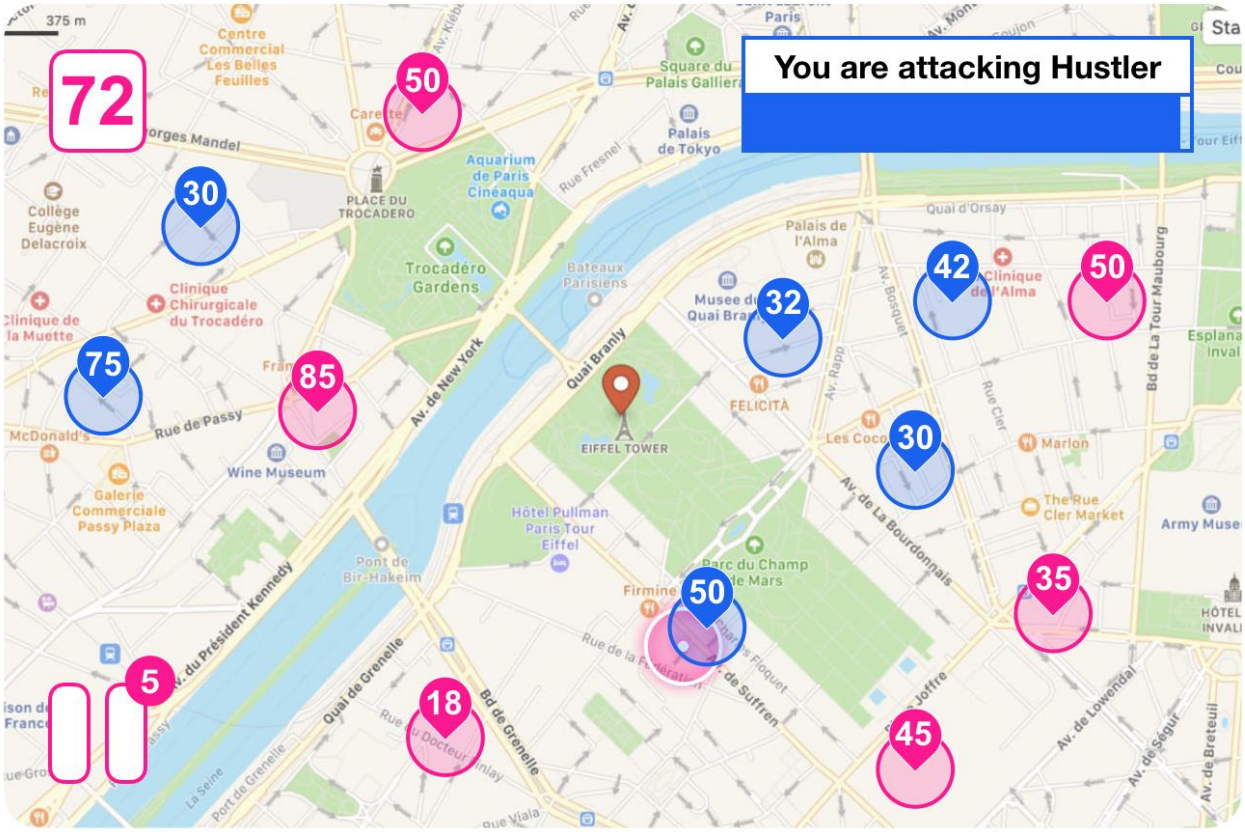

Рисунок 2.12 – Екран початку атаки

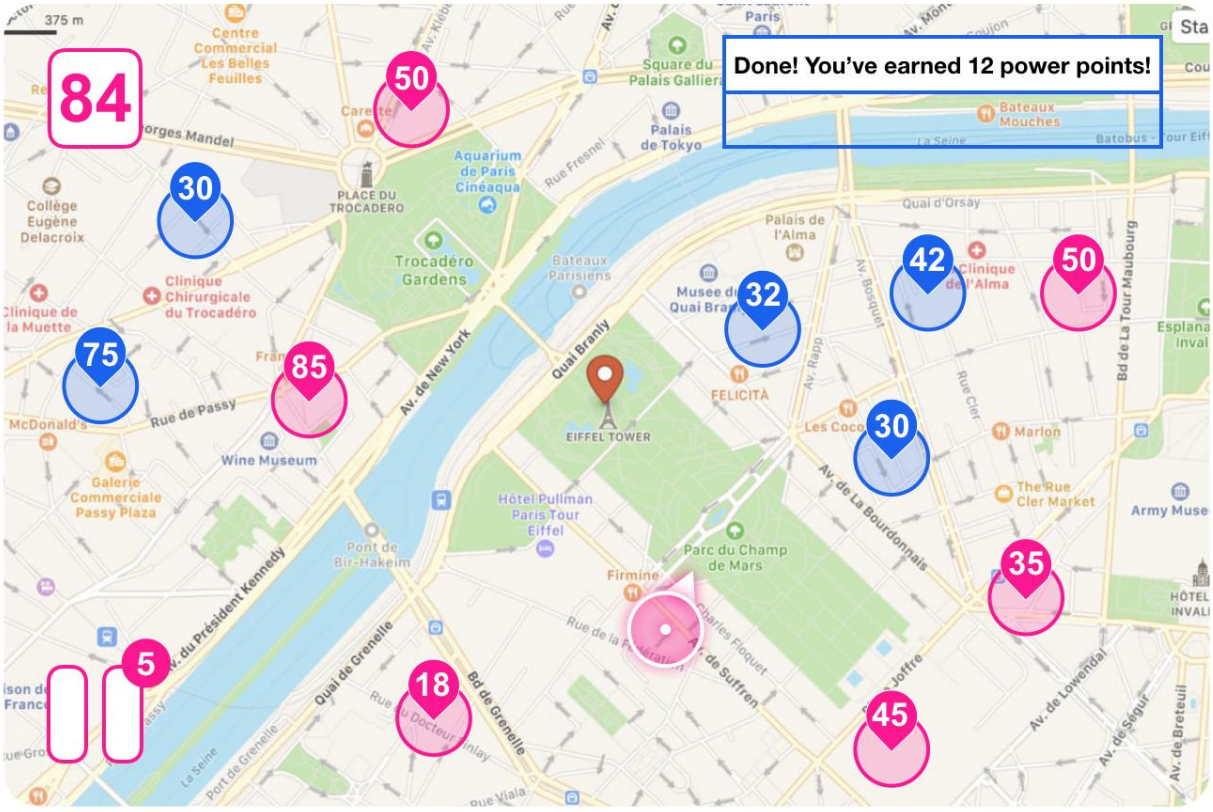

Рисунок 2.13 – Екран завершення атаки

2.4 Аналіз безпеки даних

Клієнт-серверна взаємодія відбуватиметься через протокол HTTPS, що є розширенням протоколу HTTP з підтримкою криптографічного шифрування. Він використовує криптографічні протоколи SSL та TLS.

HTTPS не є самостійним протоколом, це просто HTTP, що використовує криптографічні протоколи. HTTPS достатньо, щоб захистити дані від найрозповсюдженіших атак, і механізми iOS має його нативну підтримку. Більш того, середовище iOS за замовчуванням не дозволяє додатку надсилати запити через HTTP, і примусово вимагає HTTPS. Під час розробки, це дозволить впевнитись, що усі запити проходять через безпечний протокол, і можливі вразливості попереджені [10].

Локація користувача є найважливішою інформацію, для якої потрібно створити механізми захисту. Усі гравці в поточній грі знають, де знаходяться інші гравці, проте ці дані анонімні. Тобто справжні імена та контакти інших

гравців невідомі. Небезпека полягає в ситуації, коли одна людина може дізнатись місцезнахождення іншої без її дозволу, встановити її особистіть та почати її переслідувати.

Задля запобігання такої ситуації, локації гравців не зберігатимуться в базі даних. Отже, навіть якщо зловмисник отримає доступ до бази, він не знайде там локації. Проте, за даними в базі її можна буде вирахувати та встановити особистіть.

Справжнє ім’я та контакти людини будуть зберігатись в базі в зашифрованому вигляді. Доступ до них матимуть тільки власне гравці та адміністратори. У разі, якщо хтось з гравців звітує про небезпечну поведінку іншого, адміністратор може встановити особистіть людини, що загрожує, здобити попередження та повідомити поліцію.

Вхід у додаток відбувається через Facebook, одже вся турбота про безпечність доступу до аккаунта перекладається саме на соціальну мережу. Це є досить надійним, адже Facebook має багато інструментів безпеки, перевірених часом.

2.5 Висновки до розділу

У цьому розділі були детально розроблені сценарії клієнт-серверної взаємодії та схематично зображені за допомогою діаграм діяльності. На основі даних розробок було змодельовано взаємодію між клієнтом та сервером та розглянуто їх компоненти. Для реалізації кожної компоненти вибрані відповідні технології.

Також у рамках цього розділу було спроектовано дизайн клієнтської частити додатку у відповідності до iOS Human Interface Guidelines.

Була проаналізована безпека додатку для користувача як один із найважливіших елементів приємного користування, вибрані технології та підходи для забезпечення анонімності даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Аналізувати якість даного програмного забезпечення будемо за стандартом **ISO 9126-1**. Модель ISO 9126-1 зображена на рисунку 3.1.

Рисунок 3.1 – Модель якості ПО ISO 9126-1

Функціональність

Функціональну справність, функціональну сумісність та відповідність стандартам перевірено при тестуванні. План тестування описано в документі «Програма та методика тестування».

Точність оновлення локації – з точністю до метра. Цього цілком достатньо для потреб гри.

Безпека користування (безпека особистих даних користувача) забезпечується HTTPS з'єднанням та шифруванням інформації в базі даних.

Зручність використання

Зручніть використання перевіряється в рамках тестування зручності. Оскільки при розробці дизайну було дотримано рекомендацій Human Interface Guidelines, додаток є зручним та звичним для користувачів iOS.

Зручність супровіду

Інструменти для супровіду та аналізу описано в розділі 4. І для супровіду, і для аналізу, використано сучасні перевірені додатки.

Надійність

Система здатна опрацьовувати велику кількість різноманітних помилок та видавати коректні результати при некоректних даних.

Уся необхідна інформація зберігається на сервері, отже, якщо відбулась критична помилка з боку клієнта, він просто перезавантажиться та отримає нову актуальну інформацію від сервера.

Ефективність

Усі ігри на основі геолокації споживають багато заряду батареї. Регулювати це можна, змінюючи точність оновлення локації – метри, кілометри і т.д. Вище точність – вище споживання енергії. У цій грі принципово важлива висока точність, отже споживання енергії також буде високим. Проте, оскільки це виправдано, це можна вважати ефективним використанням ресурсів.

Використання оперативної пам’яті не перевищує 50 Мб. Це дуже мало відносно загального обсягу оперативної пам’яті пристроїв, отже пам’ять використовується достатньо ефективно.

Дані про нову локацію відправляються на сокет щосекунди. Кожні 10 секунд надходить контрольний пакет інформації. Таким чином кожну секунду клієнти отримують актуальну інформації. Проте, якщо сталася помилка, та дані перестали надходити на сокет, через 10 секунд гарантовано надійде пакет з новими даними. Цей час користувач не встигне помітити. Використання часу – ефективне.

Портативність

Встановлення відбувається через TestFlight/AppStore. Це є звичні, найдійні та зручні інструменти.

3.2 Опис процесів тестування

Тестування відбуватиметься відповідно до функціональних вимог та прописаних сценаріїв використання. Тестувальник вивчає сценарії, на основі них складає тест-кейси, формує очікуваний результат від певних дій. Задача тестування: перевірити, чи відповідають реальні результати очікуваним.

Функціональність, що була протестована у рамках виконання даного дипломного проекту:

- вхід через Facebook;
- встати в чергу на раунд;
- відображення гравців на карті;
- пауза;
- атака;
- кооперація в атаці.

Рекомендоване складання таких груп тест-кейсів: негативні, позитивні, специфічні та додаткові.

Негативні тест-кейси мають під собою перевірку системи на випадок надання їй некоректних даних. Наприклад, замість поточної локації гравця на сервер надішли дані типу String. Очікуваний результат: сервер відловлює некоректний тип даних та надсилає повідомлення про помилку.

Позитивна група тест-кейсів – вводимо коректні дані. Приклад: користувач натиснув кнопку «Увійти через Facebook», надав доступ до своїх даних. Очікувана поведінка: авторизацію успішно здійснено, відображено екран гри.

До специфічних належить такий вид кейсів, коли користувач вводить не зовсім коректні дані, проте з них можна витягнути коректні і система відпрацює без помилок. На прикладі даного проекту: на сервер прийшли 2 запити на гру від одного й того самого гравця. Це некоректно, адже 1 гравець може знаходитись тільки в одній черзі одночасно. Очікувана поведінка: сервер ігнорує запит, що надійшов пізніше, доки не надійде запит на відмову першого запиту. Таким чином, система виконала необхідні функції при введені некоректних даних.

Додаткові тест-кейси охоплюють функціональність, яка не стосується вимог безпосередньо, проте очікується від застосунку. Наприклад, користувачі iOS звикли, що коли нове вікно з'являється зліва-направо, то зникає воно зправа-наліво. Не відповідність цій поведінці не критична для функціонування системи, проте бажано її дотримуватись [11].

Модульне тестування

Модульне тестування перевіряє окремі компоненти в ізоляції методом чорної скриньки, тобто власне їх інтерфейс, без знання про внутрішню будову компонентів. Перевірена поведінка при звичайних та граничних значеннях. До кожного класу, що потребує тестування (зазвичай це viewModel – GameViewModel, LoginViewModel) – створено відповідний файл *назва класу*Tests, де, за допомогою нативного фреймворка XCTest

створено клас та функції тестування. Вони автоматично підтягуються схемою IDE та виконуються на запит розробника.

Клас тестування є нащадком класа XCTestCase та має таку структуру:

а) функція setup() – тут відбувається створення моделі, що тестується та

її налаштування під потреби тестування. Ця функція викликається перед початком виконання усіх тестів;

б) функція tearDown() – викликається після виконання всіх функцій-тестів цього класу. Тут можна проаналізувати результати тестування та «зачистити» певні дані за потреби;

в) власне, функції тестування. Функції розпізнаються IDE як функції тестування, якщо вони містять хоча б один виклик функції XCTAssert або подібних. Ця група функцій створена для порівняння очікуваних від системи результатів з реальними. Якщо всі XCTAssert повернули результат «Істина», тест вважається пройденим. Якщо хоч одна функція тестування з усього проекту повернула результат «Хибність», тестування вважається «проваленим», і проект у такому стані не може піти до користувача, доки проблема не буде вирішена.

Інтеграці□не тестування

Проект вступив в інтеграці□не тестування після того, як усі критичні дефекти в компонентах були виправлені. Інтеграці□не тестування виконувалось поступово, по мірі реалізації□ компонентів, тобто, коли був реалізовании□ новии□ компонент, перевірялась и□ого взаємодія з попередньо реалізованими. До того моменту, як певні модулі ще не були реалізовані, і□х місце заи□мали певні “заглушки”. Таким чином, можна було розпочинати інтеграці□не тестування ще до реалізації□ системи повністю.

Інтеграціне тестування проводилось мануально, на всіх рівнях абстракцій.

Найбільший рівень абстрації виділяє два основних компоненти як об’єкти інтеграційного тестування: клієнт та сервер. Клієнт містить особливу абстракцію для взаємодії з сервером – GameEngine. Оскільки саме клієнтська частина була реалізована в першу чергу, методи GameEngine повертали фейкові дані, які система сприймала як дані зі справжнього сервера. Таким чином, взаємодію цих компонентів вдалося налагодити ще до того, як сервер був реалізований.

Наступний рівень абстракції – компоненти клієнта (графічний інтерфейс, модель, бізнес-логіка, частина взаємодії з сервером) та сервера (API, база даних, бізнес-логіка).

Найнижчі рівні – інтеграція між окремими класами – наприклад, між класами LoginViewModel та LoginViewController.

Системне тестування

Системне тестування проводилось після успішного інтеграційного тествання, та було направлене на виявлення проблем в ігровому процесі, механіках гри, відображенні графіки, та перевірку відповідності гри вимогам.

Особлива увага зверталась на плавність та чіткість відображення локацій гравців обох команд на карті, на взаємодію гравця зі своєю відміткою на карті та на анімацію атаки.

Системне тестування проводилось мануально.

Тестування інтерфеи□су користувача

Виконувалось мануально після успішного проходження інтеграційого тестування. Перевірялась відповідність вимогам до графічного дизаи□ну (макету, створеного у попередньому розділі) при різних розмірах та розширеннях екрану, на різних пристроях, в портретніи□ та альбомніи□ орієнтації□, відповідність гаи□длаи□нам. Також перевірялась реакція додатку на взаємодію користувача з сенсором.

Тестування зручності

Після тестування інтерфейсу користувача проводилось тестування зручності. Для цього були залучені люди, що не стикалися з проектом до цього, та перевірена їх взаємодія з застосунком. Приклади аспектів, на які зверталась увага під час тестування:

- чи зрозумілий процес авторизації;
- чи розуміє гравець, де на карті знаходиться він, де його команда, а де

супротивники;

- чи панікує людина, коли її атакують;
- чи розуміє, як їй найкраще спланувати свою атаку, та як зберегти себе

від атаки на себе;

- чи зрозумілий механізм використання пауз.

Також був перевірении UX окремих елементів: чи достатньо великі кнопки для того, щоб попасти дотиком, чи читається текст, чи відповідають назви кнопок функціям, які вони виконують.

Автоматизоване тестування

В даному проекті є сенс автоматизувати модульні тести, тому що вони будуть виконуватись велику кількість разів (кожного разу після додавання новоі функціі до компоненту), а також модульні тести легко піддаються автоматизаціі.

Критерій проходження тестів: всі модульні тести проходять успішно, тобто всі основні функціональні можливості компонентів працюють як розраховувалось. Відсоток загальноі кількості успішних тестів має становити більше 97%, і не повинно бути жодних критичних чи сери озних помилок.

Якщо є будь-які критичні чи мажорні помилки в основніі функціональності, то тестування користувацького інтерфеи су, зручності

повинні бути призупинені. Модульне, інтеграціоне та системне тестування можна продовжувати.

Відновлення відбудеться після виправлення зазначених помилок.

Якщо з оновленням операціоної системи в ніи виникає помилка, що заважає нормальному функціонуванню основних елементів гри, тестування для поточноі версіі повністю зупиняється.

Відновлення станеться, коли розробник ОС виправить помилку.

3.3 Опис контрольного прикладу

Варіанти використання, перевірені в рамках тестування:

- вхід через Facebook (таблиця 3.1);
- почати гру (таблиця 3.2);
- пауза (таблиця 3.3);
- атака (таблиця 3.4).

Таблиця 3.3.1 – Перевірка можливості входу через Facebook

Мета тесту	Перевірка можливості входу через Facebook
Початковий стан	Відкрито додаток.
Вхідні дані	Відсутні.
Схема проведення тесту	Натиснути кнопку «Увійти через Facebook», у діалоговому вікні натиснути «ОК».
Очікуваний результат	Відображено екран початку гри з кнопками початку гри та виходу з аккаунту.
Стан програмного продукту після проведення випробувань	Користувача авторизовано в додатку.

Таблиця 3.2 – Перевірка можливості початку гри

Мета тесту	Перевірка можливості початку гри
Початковий стан	Користувача авторизовано.

Вхідні дані	Особисті дані користувача.
Схема проведення тесту	Натиснути кнопку «Встати в чергу»; дочекатись повідомлення, що необхідна кількість гравців набрана та гру розпочато.
Очікуваний результат	Відображено екран гри: локації всіх користувачів зображені маркерами, колір яких відповідає кольору їх команд, відображено рівень їх сили та кнопку паузи.
Стан програмного продукту після проведення випробувань	Гру розпочато.

Таблиця 3.3 – Перевірка можливості паузи гри

Мета тесту	Перевірка можливості паузи гри
Початковий стан	Гру розпочато.
Вхідні дані	Кількість пауз, що залишились у користувача на сьогодні.
Схема проведення тесту	Натиснути кнопку «Пауза».
Очікуваний результат	Відображено екран паузи: кнопки повернутися до гри та вийти з гри.
Стан програмного продукту після проведення випробувань	Для даного користувача гру поставлено на паузу; його локацію більше не бачать інші користувачі, він також не бачить локацію інших.

Таблиця 3.4 – Перевірка можливості атаки

Мета тесту	Перевірка можливості атаки
Початковий стан	Гру розпочато.
Вхідні дані	Локація клієнта-нападника та клієнта-жертви.
Схема проведення тесту	Підійти до супротивника з меншим рівнем сили

	на відстань 50 метрів або менше; знаходитись у цій зоні протягом 5 хвилин, доки не надійде повідомлення про успішність атаки.
Очікуваний результат	Гравця-жертву більше не відображено на карті, гравцю-нападнику нараховано 12 балів сили.
Стан програмного продукту після проведення випробувань	Гравець-жертва вибув з гри, гравцю-нападнику нараховано бали сили.

3.4 Висновки по розділу

У цьому розділі були виділені функціональності проекту, що мають бути протестовані. На практиці, вони співпали з вимогами, тож проект було протестовано на відповідність усім зазначеним функціональним вимогам.

Описано загальні підходи до тестування та виділені необхідні групи тест-кейсів. Розроблено тест план.

У рамках розділу надано загальний опис методик тестування з конкретними прикладами та обґрунтуванням їх використання. Встановлені критерії прохождення тестів.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання сервера використовується утиліта The Perfect Assistant.

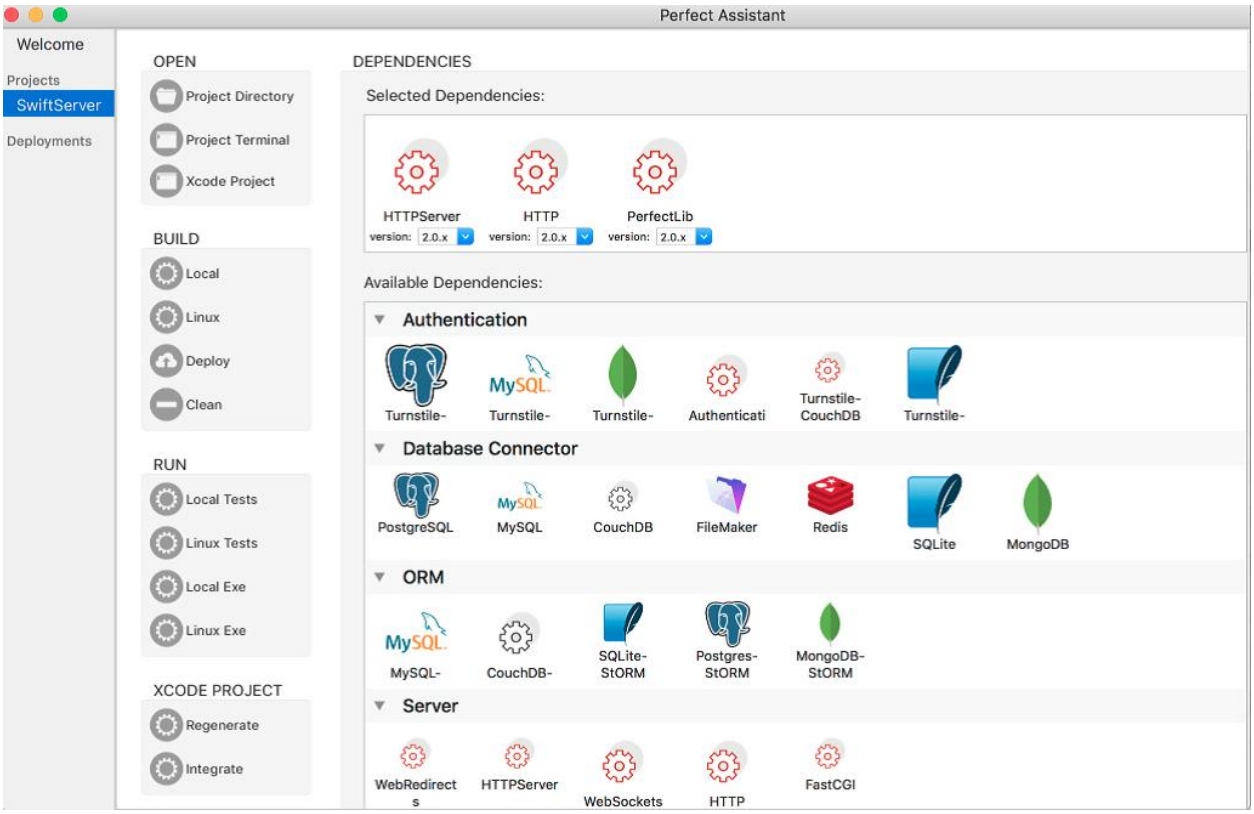

Рисунок 4.1 – Інтерфейс The Perfect Assistant

Він є інтуїтивно зрозумілим і дозволяє вирішити всі задачі по розгортанню та запуску натиском на кілька кнопок [12].

Для встановлення клієнтського застосунку потрібно мати операційну систему iOS версії 11.0 або вище. iOS – операційна система закритого типу, тож заради безпеки є тільки 2 шляхи встановити додаток на неї (без взлому системи):

– магазин AppStore;

– TestFlight – офіційний застосунок від Apple, що дозволяє розробникам тестувати свої наробки до завантаження їх до AppStore.

Для завантаження додатків у AppStore потрібно мати ліцензію розробника, що коштує 100$ на рік. У рамках дипломного проекту такий варіант не розглядається.

Наразі застосунок можна завантажити через TestFlight, маючи попереднє запрошення.

Також для авторизації необхідний функціонуючий Facebook-сервер.

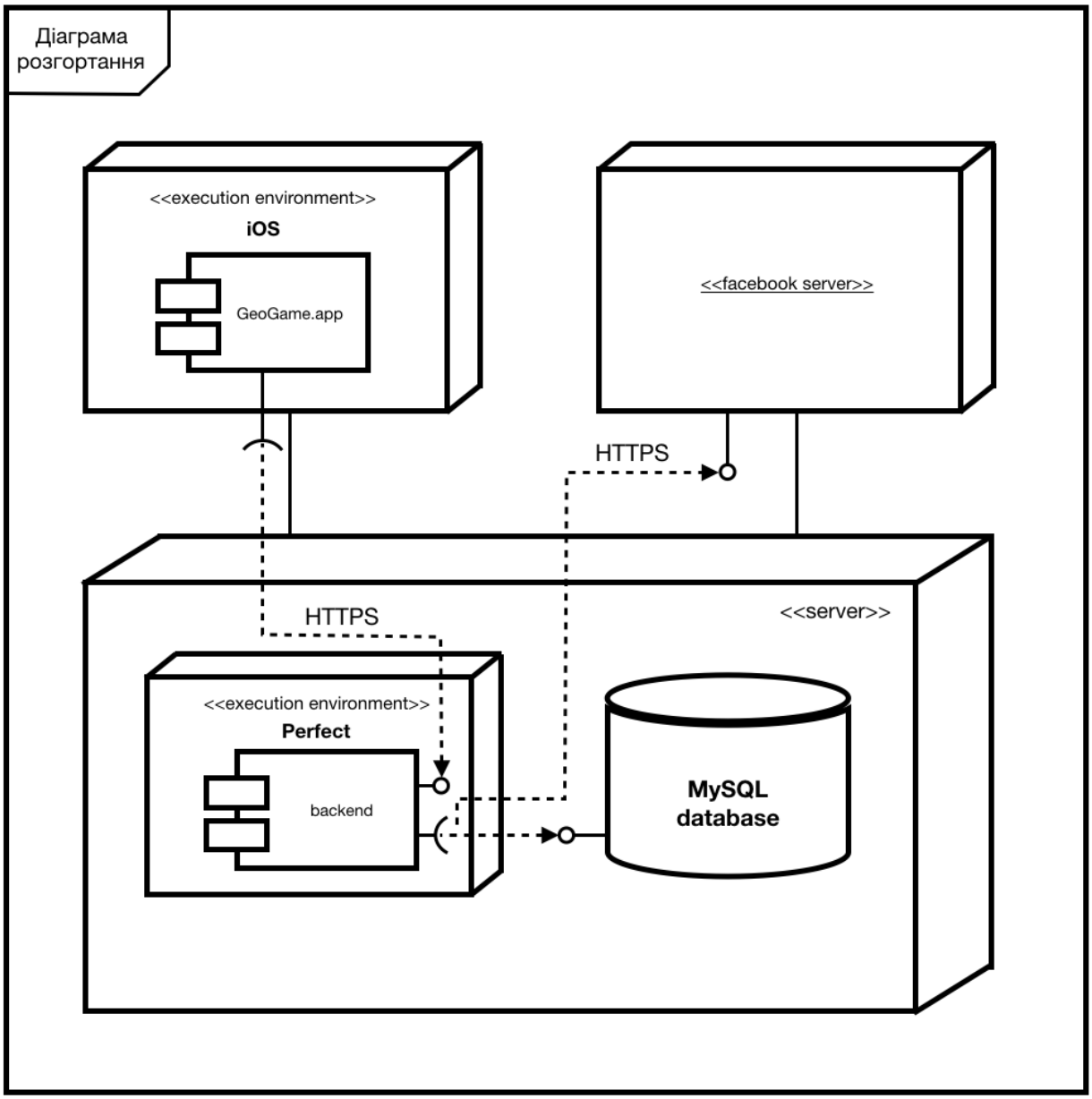

Рисунок 4.2 – Діаграма розгортання

4.2 Робота з програмним забезпеченням

Детальний покроковий опис дій щодо користування iOS додатком наведено в документі «Керівництві користувача». Наведемо опис основних етапів роботи.

Авторизація (передумова – користувач відкрив додаток):

а) натиснути «Вхід через Facebook»;

б) у діалоговому вікні натиснути «Дозволити»;

Початок гри (передумова – користувача авторизовано):

а) натиснути кнопку «Встати в чергу»;

б) дочекатись повідомлення, що необхідна кількість гравців набрана та гру розпочато.

Атака (передумова – гру розпочато):

а) підійти до супротивника з меншим рівнем сили на відстань 50 метрів

або менше;

б) знаходитись у цій зоні протягом 5 хвилин, доки не надійде повідомлення про успішність атаки.

Ухиляння від атаки (передумова – гравця атаковано):

а) відійти від гравця, що атакує, на відстань більше 50 метрів;

б) альтернативний варіант: вийти з гри;

Поставити на паузу (передумова – гру розпочато):

а) натиснути кнопку «Пауза»;

б) якщо пауз не залишилось АБО якщо зараз гравця атакують, виникає діалогове вікно з попередженням, що пауза означатиме вихід з гри. Якщо цей варіант влаштовує, натиснути «ОК». Якщо ні, натиснути «Скасувати», і гра продовжиться;

в) якщо пауз залишилось більше 0, гра ставиться на паузу, локація перестає надаватись.

Вихід з гри (передумова – гру розпочато):

а) натиснути кнопку «Пауза»;

б) натиснути кнопку «Вихід з гри»;

в) у діалоговому вікні вибрати «Так».

4.3 Супровід програмного забезпечення

Для аналізу помилок використовується сервіс DevMate. Він збирає репорти про помилки та ексепшини на стороні клієнта, а також збирає статистику використання, інсталювань і так далі. Через DevMate також зручно збирати зворотній зв’язок від користувачів.

Рисунок 4.3 – Аналітика в DevMate

4.4 Висновки по розділу

У цьому розділі були сформовані вимоги для розгортання, підтримки та користування створеним програмним забезпеченням.

Визначено необхідний інструмент для запуску та розгортання серверної частини – The Perfect Assistant.

Описано шляхи завантаження додатку на iOS та вимоги для функціонування додатку.

Створено наочну діаграму розгортання, що відображає зв’язки між елементами.

Створено інструкцію користувача та наведено опис основних етапів роботи із iOS додатком.

Визначено програмне забезпечення для детекції та аналізу помилок, та збору аналітики.

Таким чином, система на даному етапі готова до запуску та повноцінного функціонування.

5 ВИСНОВКИ

У ході виконання дипломної роботи було створено готовий до використання програмний продукт – багатокористувацьку мережеву гру на основі геолокації.

Першим етапом був аналіз предметної області та конкурентів, задля визначення свого місця в цій області. Під час аналізу було виявлено сильні та слабкі сторони конкурентів, і на основі них виявлені потенційні переваги даного застосунку.

Були розроблені функціональні вимоги та створено детальний опис сценаріїв взаємодії користувача з клієнтською частиною застосунку, а також описано взаємодію клієнтської та серверної частини. Визначено мету розробки.

Наступний етап полягав у розробці архітектури, створенні компонентів та дизайну. Далі застосунок був протестований та підготовлений до запуску.

Отриманий застосунок відповідає всім заявленим вимогам. Окрім того, гра є цікавою та виконує свій мотивуючий аспект – гравці прагнуть переміщатися містом аби атакувати та ухилятись від атак. Інтерфейс є інтуїтивно зрозумілим. Запити до сервера надходять швидко, анімації є плавними. Анонімність користувачів збережено.

Таким чином, додаток виконує своє призначення – перенесення гри в реальний світ та поєднання розваг із користю.

ПЕРЕЛІК ПОСИЛАНЬ

1) How the Geolocation works [Електронний ресурс]: (Стаття) / Users Insights. – Електрон. Дан. (1 файл). – 2015. – Режим доступу: https://usersinsights.com/how-the-geolocation-works/. – Назва з екрана.

2) Pokémon Go [Електронний ресурс]: (Веб-сторінка) / Pokémon Go. – Електрон. Дан. (1 файл). – 2019. – Режим доступу: https://www.pokemongo.com/en-us/. – Назва з екрана.

3) Жанри комп'ютерних ігор [Електронний ресурс]: (Стаття) / Games is art. – Електрон. Дан. (1 файл). – 2014. – Режим доступу: http://gamesisart.ru/janr.html. – Назва з екрана.

4) CoreLocation [Електронний ресурс]: (Стаття) / Apple Developer. – Електрон. Дан. (1 файл). – 2019. – Режим доступу: https://developer.apple.com/documentation/corelocation. – Назва з екрана.

5) MapKit [Електронний ресурс]: (Стаття) / Apple Developer. – Електрон. Дан.
(1 файл). – 2019. – Режим доступу: https://developer.apple.com/documentation/mapkit. – Назва з екрана.

6) Ingress [Електронний ресурс]: (Веб-сторінка) / Ingress. – Електрон. Дан. (1 файл). – 2019. – Режим доступу: https://www.ingress.com. – Назва з екрана.

7) UML Activity Diagram Tutorial [Електронний ресурс]: (Стаття) / Lucid Chart. – Електрон. Дан. (1 файл). – 2016. – Режим доступу: https://www.lucidchart.com/pages/uml-activity-diagram. – Назва з екрана.

8) MVVM [Електронний ресурс]: (Стаття) / Metanit. – Електрон. Дан. (1 файл). – 2016. – Режим доступу: https://metanit.com/sharp/wpf/22.1.php. – Назва з екрана.

9) Human Interface Guidelines [Електронний ресурс]: (Веб-сторінка) / Apple

Developer. – Електрон. Дан. (1 файл). – 2019. – Режим доступу: https://developer.apple.com/design/human-interface-guidelines/. – Назва з екрана.

10) Відмінності HTTP та HTTPS [Електронний ресурс]: (Стаття) / Users Insights. – Електрон. Дан. (1 файл). – 2014. – Режим доступу: https://ssl.com.ua/blog/http-vs-https/. – Назва з екрана.

11) Загальні підходи при створенні тест плану [Електронний ресурс]: (Стаття) / Users Insights. – Електрон. Дан. (1 файл). – 2015. – Режим доступу: https://www.softwaretestingmaterial.com/test-plan-template/. – Назва з екрана.

12) The Perfect Assistant [Електронний ресурс]: (Стаття) / Perfect. – Електрон. Дан. (1 файл). – 2019. – Режим доступу: https://perfect.org/en/assistant/. – Назва з екрана.

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ О.А. Павлов
“___” _____ 2019 р.

Мережеве ігрове програмне забезпечення з використанням геолокації

Опис програми

КПІ.ІП-5120.045490.03.13

“ПОГОДЖЕНО”
Керівник проекту:
_____ Г.В. Ісаченко

Нормоконтроль:
_____ М.М. Головченко

Виконавець:
_____А.С. Старченко

Київ – 2019 року

Тексти програмного коду

Мережеве ігрове програмне забезпечення з використанням геолокації

(Найменування програми (документа))

CD-R

(Вид носія даних)

14 арк, 232 Кб

(Обсяг програми (документа) , арк., Кб)

Київ - 2019

```
class PlayerMapMarker: GMSMarker {

    let player: Player

    init(player: Player) {
        self.player = player
        super.init()
        self.position = player.location
    }
}

class PlayerMapMarkerView: UIView {

    let player: Player

    private let infoLabel = UILabel()
    private let pulsator = Pulsator()

    private let infoView = UIView()

    init(player: Player) {
        self.player = player
        super.init(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
        configureUI()
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    private func configureUI() {
        backgroundColor = .clear

        addSubview(infoView)
        let bgColor: UIColor = player.team == .green ? .blue : .magenta
        infoView.backgroundColor = bgColor.withAlphaComponent(0.2)
        infoView.layer.borderColor = bgColor.cgColor
        infoView.layer.borderWidth = 3
        infoView.layer.cornerRadius = 20
        infoView.snp.makeConstraints {
            $0.center.equalToSuperview()
            $0.size.equalTo(40)
        }

        infoView.addSubview(infoLabel)
        infoLabel.textAlignment = .center
        infoLabel.numberOfLines = 2
        infoLabel.font = UIFont.systemFont(ofSize: 10)
        infoLabel.text = "\(player.name) \n\(player.power)"
        infoLabel.snp.makeConstraints {
            $0.edges.equalToSuperview()
        }

        layer.addSublayer(pulsator)
        pulsator.radius = 50
        pulsator.numPulse = 3
        pulsator.backgroundColor = bgColor.cgColor
        pulsator.frame = CGRect(x: 50, y: 50, width: 0, height: 0)
    }

    func togglePulsator() {
        pulsator.isPulsating ? pulsator.stop() : pulsator.start()
    }
```

```
}
class GameViewController: UIViewController {

    let locationManager = CLLocationManager()
    let regionRadius: CLLocationDistance = 600

    var timer: Timer?

    let engine = GameEngine()

    var markers: [PlayerMapMarker] = []

    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        mapView.delegate = self
        setupLocationManager()
        updatePlayers()

        DispatchQueue.main.asyncAfter(deadline: .now() + 2.0) {
            self.moveMarkers()
        }
    }

    private func setupLocationManager() {
        locationManager.requestAlwaysAuthorization()
        locationManager.requestWhenInUseAuthorization()

        if CLLocationManager.locationServicesEnabled() {
            locationManager.delegate = self
            locationManager.desiredAccuracy = kCLLocationAccuracyBestForNavigation
            locationManager.startUpdatingLocation()
            locationManager.startUpdatingHeading()
        }
    }

    func updatePlayers() {
        engine.onPlayersUpdate { players in
            self.draw(players: players)
        }
    }

    func draw(players: [Player]) {
        players.forEach { player in

            let marker = PlayerMapMarker(player: player)
            let markerView = PlayerMapMarkerView(player: player)

            marker.iconView = markerView
            marker.tracksViewChanges = true
            marker.map = mapView

            markers.append(marker)
        }
    }

    func centerMapOnLocation(location: CLLocation) {
        let camera = GMSCameraPosition(target: location.coordinate, zoom: 16)
        mapView.animate(to: camera)
    }
}
```

```
extension GameViewController: CLLocationManagerDelegate {

    func makePlayer(with location: CLLocationCoordinate2D) -> Player {
        return Player(team: .red, name: "Looooooola", id: 6, location: location,
power: 50)
    }

    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
[CLLocation]) {
        guard let location = manager.location else { return }

        let mockedLocation = CLLocation(latitude: 50.442029, longitude: 30.4410228)
        centerMapOnLocation(location: mockedLocation)

        let player = makePlayer(with: mockedLocation.coordinate)
        engine.synchronize(player: player)
    }
}

extension GameViewController: GMSMapViewDelegate {

    func mapView(_ mapView: GMSMapView, didTap marker: GMSMarker) -> Bool {
        guard let playerMarker = marker as? PlayerMapMarker,
            let markerView = playerMarker.iconView as? PlayerMapMarkerView
        else { return false }

        markerView.togglePulsator()
        return true
    }
}

extension GameViewController {

    private func moveMarkers() {
        for marker in markers {
            var position = marker.position

            let topMove = Bool.random()
            let rightMove = Bool.random()

            if topMove {
                position.longitude -= 0.002
            } else {
                position.longitude += 0.002
            }

            if rightMove {
                position.latitude += 0.0001
            } else {
                position.latitude -= 0.0001
            }

            marker.position = position
        }
    }
}

struct Player {

    let team: Team
    let name: String
    let id: Int

    let location: CLLocationCoordinate2D
```

```
    let power: Int
}

enum Team {

    case green, red
}
class ViewController: UIViewController, FBSDKLoginButtonDelegate {

    func loginButton(_ loginButton: FBSDKLoginButton!, didCompleteWith result:
FBSDKLoginManagerLoginResult!, error: Error!) {

    }

    func loginButtonDidLogOut(_ loginButton: FBSDKLoginButton!) {

    }


    override func viewDidLoad() {
        super.viewDidLoad()

        let vc = MapViewController()
        self.navigationController?.pushViewController(vc, animated: true)

        let loginButton = FBSDKLoginButton()
        loginButton.center = view.center
        loginButton.readPermissions = ["email"]
        loginButton.delegate = self
        view.addSubview(loginButton)

        if let token = FBSDKAccessToken.current() {

        }
    }
}
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        FBSDKApplicationDelegate.sharedInstance().application(application,
didFinishLaunchingWithOptions: launchOptions)
        return true
    }

    func application(_ application: UIApplication, open url: URL, sourceApplication:
String?, annotation: Any) -> Bool {
        return FBSDKApplicationDelegate.sharedInstance().application(application,
open: url, sourceApplication: sourceApplication, annotation: annotation)
    }
}
class DataManager {

    private var ways: [Way] = []
    private var timeForWay: Int = 0


    private init() {}
    static let instance = DataManager()
```

```
    func getWays () -> [Way] {
        return ways
    }

    func getTime() -> Int {
        return timeForWay
    }

    func readInfoFromJson () {


        let localURL = Bundle.main.url(forResource: "Kiev", withExtension: "json")

        let fileData: Data?
        do {
            fileData = try Data(contentsOf: localURL!)
        } catch {
            print("Read data error")
            return
        }

        do {
            if let json = try JSONSerialization.jsonObject(with: fileData!) as?
[String: Any],

                let lines = json["lines"] as? [[String: Any]] {
                    for line in lines {
                        let color = line["color"] as! String
                        let name = line["name"] as! String

                        var thisLineStations: [Station] = []
                        if let stations = line["stations"] as? [[String: Any]] {
                            for station in stations {
                                let lat = station["lat"] as! Double
                                let lon = station["lon"] as! Double
                                let statName = station["name"] as! String
                                let timeNext = station["timeNext"] as! Int
                                let timePrev = station["timePrev"] as! Int
                                let id = station["id"] as! Int

                                let stat = Station(stationName: statName, stationId:
id, x: lat, y: lon, timeN: timeNext, timeP: timePrev)
                                thisLineStations.append(stat)
                             }
                        }

                        ways.append(Way(myStations: thisLineStations, myName: name,
myColor: color))
                    }
                }
        } catch {
                print("Error deserializing JSON: \(error)")
        }

    }


    func initWays () {

        readInfoFromJson()

        if let station1 = ways[0].getStationByName(name: "Zoloti vorota"), let
station2 = ways[2].getStationByName(name: "Teatralna") {
```

```
            ways[0].transfers.append([station1, station2])
            ways[2].transfers.append([station2, station1])
        }

        if let station1 = ways[0].getStationByName(name: "Palats sportu"), let
station2 = ways[1].getStationByName(name: "Lva Tolstogo Square") {
            ways[0].transfers.append([station1, station2])
            ways[1].transfers.append([station2, station1])
        }

        if let station1 = ways[2].getStationByName(name: "Khreshchatyk"), let
station2 = ways[1].getStationByName(name:  "Maidan Nezalaznosti") {
            ways[2].transfers.append([station1, station2])
            ways[1].transfers.append([station2, station1])
        }


    }


    func buildWay (from: String, to: String) -> [Way] {

        var fromWay: Way?
        var toWay: Way?

        var fromStation: Station?
        var toStation: Station?

        var newWay: [Way] = []

        for way in ways {
            if let station = way.getStationByName(name: from) {
                fromWay = way
                fromStation = station
            }

            if let station = way.getStationByName(name: to) {
                toWay = way
                toStation = station
            }
        }


        if fromWay?.name == toWay?.name {

            if let way = fromWay, let fromSt = fromStation, let toSt = toStation,
let color = fromWay?.color {
                newWay.append(Way(myStations: findPathInOneWay(fromStation: fromSt,
toStation: toSt, myWay: way), myName: "", myColor: color))
            }


        } else {

            if let transfers = fromWay?.transfers {
                for item in transfers {
                    if let station1 = fromWay?.getStationByName(name: item[0].name),
                        let station2 = toWay?.getStationByName(name: item[1].name),
                        station2.id != -1,
                        let wayT = toWay,
                        let wayFr = fromWay,
                        let color1 = fromWay?.color,
                        let color2 = toWay?.color {
```

```
                        newWay.append(Way(myStations: findPathInOneWay(fromStation:
fromStation!, toStation: station1, myWay: wayFr), myName: "", myColor: color1))

                        newWay.append(Way(myStations: findPathInOneWay(fromStation:
station2, toStation: toStation!, myWay: wayT), myName: "", myColor: color2))


                        break

                        }
                }
            }
        }







        return newWay

    }

    func findPathInOneWay (fromStation: Station, toStation: Station, myWay: Way) ->
[Station] {
        let fromId = fromStation.id
        let toId = toStation.id
        var way: [Station] = []

        var currentStation = Station(with: fromStation)

        if fromId > toId {
            for _ in toId...fromId {
                let station = Station(with: currentStation)
                way.append(Station(with: station))
                timeForWay += station.timeToPrevStation
                if var prevStat = station.prev {
                    prevStat = Station(with: prevStat)
                    currentStation = Station(with: prevStat)
                }
            }

        } else {
            for _ in fromId...toId {
                let station = Station(with: currentStation)
                way.append(Station(with: station))
                timeForWay += station.timeToNextStation
                if var nextStat = station.next {
                    nextStat = Station(with: nextStat)
                    currentStation = Station(with: nextStat)
                }
            }
        }


        return way
    }


    func destroyWay() {

        timeForWay =  0
    }
```

```
}
class ViewController: UIViewController {

    @IBOutlet weak var toTextField: UITextField!
    @IBOutlet weak var fromTextField: UITextField!

    @IBOutlet weak var metroMap: MKMapView!
    @IBOutlet weak var wayText: UITextView!


    var wayPolyline: CustomPolyline?
    var waySelectingPolyline: CustomPolyline?

    var wayPolylines: [CustomPolyline] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        toTextField.delegate = self
        fromTextField.delegate = self
        metroMap.delegate = self

        DataManager.instance.initWays()

        wayText.isHidden = true
        wayText.layer.cornerRadius = 4.0

        fromTextField.tag = 0
        toTextField.tag = 1

        showStations()

        let location = CLLocationCoordinate2DMake(50.444850, 30.516071)
        let span = MKCoordinateSpanMake(0.2, 0.2)
        let region = MKCoordinateRegion(center: location, span: span)
        metroMap.setRegion(region, animated: true)

    }


    func showStations () {
        for item in DataManager.instance.getWays() {

            for transfer in item.transfers {
                var crossingPoints: [CLLocationCoordinate2D] =
[CLLocationCoordinate2D]()
                for station in transfer {
                    let annotation = station.annotation
                    crossingPoints.append(annotation.coordinate)
                }
                let polyline = CustomPolyline(coordinates: crossingPoints, count:
crossingPoints.count)
                polyline.color = "#00000080"
                polyline.width = 3.0
                metroMap.add(polyline)
            }

            let stations = item.stations
            var points: [CLLocationCoordinate2D] = [CLLocationCoordinate2D]()


            for station in stations {

                let annotation = station.annotation
```

```
                points.append(annotation.coordinate)
                metroMap.addAnnotation(annotation)


            }

            let polyline = CustomPolyline(coordinates: points, count: points.count)
            polyline.color = item.color
            metroMap.add(polyline)




            for coordinate in points {
                var coordinates: [CLLocationCoordinate2D] =
[CLLocationCoordinate2D]()
                coordinates.append(coordinate)
                let pol = CustomPolyline(coordinates: coordinates, count:
coordinates.count)
                pol.width = 10.0
                pol.color = item.color
                metroMap.add(pol)
            }
        }
    }

    func setTextOfWay (ways: [Way]) {
        wayText.text = ""
        var textOfWay: NSMutableAttributedString?
        textOfWay = NSMutableAttributedString(string: "Ride \(ways[0].stations.count
- 1) stations to ", attributes: [:])

        let nameOfStation = Utils.getAttributedText(inputText:
"\(ways[0].stations[ways[0].stations.count - 1].name) station. ", location: 0,
length: ways[0].stations[ways[0].stations.count - 1].name.characters.count, color:
UIColor(hexString: ways[0].color)!)

        textOfWay?.append(nameOfStation)

        if ways.count == 1 {
            let addText = NSMutableAttributedString(string: "You do not need to
change the train. ", attributes: [:])
            textOfWay?.append(addText)
        } else {
            var addText = NSMutableAttributedString(string: "Then cross to ",
attributes: [:])
            textOfWay?.append(addText)
            addText = Utils.getAttributedText(inputText:
"\(ways[1].stations[0].name) station to change line and ride
\(ways[1].stations.count) stations ", location: 0, length:
ways[1].stations[0].name.characters.count, color: UIColor(hexString:
ways[1].color)!)
            textOfWay?.append(addText)

            addText = Utils.getAttributedText(inputText:
"\(ways[1].stations[ways[1].stations.count - 1].name) station. ", location: 0,
length: ways[1].stations[ways[1].stations.count - 1].name.characters.count, color:
UIColor(hexString: ways[1].color)!)
            textOfWay?.append(addText)
        }

        let timeText = NSMutableAttributedString(string: "It will take near
\(DataManager.instance.getTime()) minutes. ", attributes: [:])
```

```
            textOfWay?.append(timeText)
            wayText.attributedText = textOfWay
        }


        func makePath () {

            deleteOldWay()

            let toName = toTextField.text ?? ""
            let fromName = fromTextField.text ?? ""

            let ways = DataManager.instance.buildWay(from: toName, to: fromName)

            var points: [CLLocationCoordinate2D] = [CLLocationCoordinate2D]()

            for way in ways {
                points = []
                for item in way.stations {
                    let annotation = item.annotation
                    points.append(annotation.coordinate)

                }

                waySelectingPolyline = CustomPolyline(coordinates: points, count:
points.count)
                waySelectingPolyline?.color = "#000000ff"
                waySelectingPolyline?.width = 8.0
                if let polyline = waySelectingPolyline {
                    metroMap.add(polyline)
                    wayPolylines.append(polyline)
                }

                for coordinate in points {
                    var coordinates: [CLLocationCoordinate2D] =
[CLLocationCoordinate2D]()
                    coordinates.append(coordinate)
                    let polBlack = CustomPolyline(coordinates: coordinates, count:
coordinates.count)
                    polBlack.width = 13.0
                    polBlack.color = "#000000ff"
                    metroMap.add(polBlack)
                    wayPolylines.append(polBlack)

                    let polColor = CustomPolyline(coordinates: coordinates, count:
coordinates.count)
                    polColor.width = 10.0
                    polColor.color = way.color
                    metroMap.add(polColor)
                    wayPolylines.append(polColor)

                }

                wayPolyline = CustomPolyline(coordinates: points, count: points.count)
                wayPolyline?.color = way.color
                wayPolyline?.width = 4.0
                if let polyline = wayPolyline {
                    metroMap.add(polyline)
                    wayPolylines.append(polyline)
                }

            }
```

```
            setTextOfWay(ways: ways)
            metroMap.reloadInputViews()
            wayText.isHidden = false

        }


        func deleteOldWay () {
            DataManager.instance.destroyWay()

            for item in wayPolylines {
                metroMap.remove(item)
            }
        }
    }

    extension ViewController: MKMapViewDelegate {

        func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) ->
    MKOverlayRenderer {
            if overlay is CustomPolyline {
                let myLine: CustomPolyline = overlay as! CustomPolyline
                let renderer1 = MKPolylineRenderer(overlay: overlay)
                renderer1.strokeColor = UIColor(hexString: myLine.color)
                renderer1.lineWidth = myLine.width

                return renderer1
            }

            return MKOverlayRenderer()
        }



        func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) ->
    MKAnnotationView? {
            guard !(annotation is MKUserLocation) else {
                return nil
            }

            let annotationIdentifier = "Identifier"
            var annotationView: MKAnnotationView?
            if let dequeuedAnnotationView =
    mapView.dequeueReusableAnnotationView(withIdentifier: annotationIdentifier) {
                annotationView = dequeuedAnnotationView
                annotationView?.annotation = annotation
            }
            else {
                annotationView = MKAnnotationView(annotation: annotation,
    reuseIdentifier: annotationIdentifier)
            }

            if let annotationView = annotationView {
                annotationView.canShowCallout = true

                for way in DataManager.instance.getWays() {
                    if let title = annotation.title as? String {
                        if way.getStationByName(name: title) != nil {
                            let image = UIImage(named: way.name)?.alpha(0)
                            annotationView.image = image
                        }
                    }
                }
            }
```

```
            return annotationView
        }
    }

    extension ViewController: UITextFieldDelegate {

        func textFieldDidBeginEditing(_ textField: UITextField) {
            textField.resignFirstResponder()

            let transitionObj = TransitionObject(text: textField.text ?? "",
    textFieldTag: textField.tag)
            performSegue(withIdentifier: Utils.TableViewSegue, sender: transitionObj)
        }


        override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

            if segue.identifier == Utils.TableViewSegue {
                let destination = segue.destination as! TableViewController
                let transitionObj = sender as? TransitionObject

                destination.searchText = transitionObj?.text
                destination.textFieldTag = transitionObj?.textFieldTag
                destination.delegate = self
            }

        }
    }

    extension ViewController: MyTableViewDelegate {
        func setData(_ data: String, to tag: Int) {

            switch tag {
            case 0:
                fromTextField.text = data
            case 1:
                toTextField.text = data
            default:
                break
            }

            if fromTextField.text == toTextField.text {
                wayText.text = ""
                wayText.isHidden = false
                deleteOldWay()
                wayText.text.append("You are already here.")
            } else if fromTextField.text != "" && toTextField.text != "" {
               makePath()
            }
        }
    }

    class CustomPolyline: MKPolyline {

        var color = "#00000030"
        var width: CGFloat = 4.0
    }
```

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ О.А. Павлов
“___” _____ 2019 р.

МЕРЕЖЕВЕ ІГРОВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ГЕОЛОКАЦІЇ

Програма та методика тестування

КПІ.ІП-5120.045490.04.51

“ПОГОДЖЕНО”
Керівник проекту:
_____ Г.В. Ісаченко

Виконавець:
Нормоконтроль: _____А.С. Старченко
_____ М.М. Головченко

Київ – 2019 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАНЬ

Мережеве ігрове програмне забезпечення на основі геолокації, що є клієнт-серверним застосунком на iOS, створеним мовою Swift за допомогою IDE XCode з використанням фреймворків CocoaTouch, CoreLocation, MapKit та Perfect. Для бази даних використовувалась мова MySQL.

2 МЕТА ТЕСТУВАННЯ

Мета тестування полягає у перевірці якості створеного програмного продукту; перевірці, чи відповідає продукт поставленим вимогам. Задля цього буде перевірений такий функціонал:

- вхід через Facebook;
- встати в чергу на раунд;
- відображення гравців на карті;
- пауза;
- атака;
- кооперація в атаці.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування проводиться методом чорної скриньки.

Модульне тестування

Модульне тестування перевіряє окремі компоненти в ізоляції методом чорної скриньки, тобто власне їх інтерфейс, без знання про внутрішню будову компонентів. Перевірена поведінка при звичайних та граничних значеннях.

Інтеграційне тестування

Проект вступив в інтеграційне тестування після того, як усі критичні дефекти в компонентах були виправлені. Інтеграційне тестування виконувалось поступово, по мірі реалізації компонентів, тобто, коли був реалізований новий компонент, перевірялась його взаємодія з попередньо реалізованими. До того моменту, як певні модулі ще не були реалізовані, їх місце займали певні “заглушки”. Таким чином, можна було розпочинати інтеграційне тестування ще до реалізації системи повністю.

Інтеграціне тестування проводилось мануально, на всіх рівнях абстракцій.

Найбільший рівень абстракції виділяє два основних компоненти як об’єкти інтеграційного тестування: клієнт та сервер. Клієнт містить особливу абстракцію для взаємодії з сервером – GameEngine. Оскільки саме клієнтська частина була реалізована в першу чергу, методи GameEngine повертали фейкові дані, які система сприймала як дані зі справжнього сервера. Таким чином, взаємодію цих компонентів вдалося налагодити ще до того, як сервер був реалізований.

Наступний рівень абстракції – компоненти клієнта (графічний інтерфейс, модель, бізнес-логіка, частина взаємодії з сервером) та сервера (API, база даних, бізнес-логіка).

Найнижчі рівні – інтеграція між окремими класами – наприклад, між класами LoginViewModel та LoginViewController.

Системне тестування

Системне тестування проводилось після успішного інтеграційного тествання, та було направлене на виявлення проблем в ігровому процесі, механіках гри, відображенні графіки, та перевірку відповідності гри вимогам.

Особлива увага зверталась на плавність та чіткість відображення локацій гравців обох команд на карті, на взаємодію гравця зі своєю відміткою на карті та на анімацію атаки.

Системне тестування проводилось мануально.

Тестування інтерфейсу користувача

Виконувалось мануально після успішного проходження інтеграційого тестування. Перевірялась відповідність вимогам до графічного дизайну (макету, створеного у попередньому розділі) при різних розмірах та розширеннях екрану, на різних пристроях, в портретній та альбомній орієнтації, відповідність гайдлайнам. Також перевірялась реакція додатку на взаємодію користувача з сенсором.

Тестування зручності

Після тестування інтерфейсу користувача проводилось тестування зручності. Для цього були залучені люди, що не стикалися з проектом до цього, та перевірена їх взаємодія з застосунком. Приклади аспектів, на які зверталась увага під час тестування:

– чи зрозумілий процес авторизації;

– чи розуміє гравець, де на карті знаходиться він, де його команда, а де супротивники;

– чи панікує людина, коли її атакують;

– чи розуміє, як їй найкраще спланувати свою атаку, та як зберегти себе від атаки на себе;

– чи зрозумілий механізм використання пауз.

Також був перевірений UX окремих елементів: чи достатньо великі кнопки для того, щоб попасти дотиком, чи читається текст, чи відповідають назви кнопок функціям, які вони виконують.

Автоматизоване тестування

В даному проекті є сенс автоматизувати модульні тести, тому що вони будуть виконуватись велику кількість разів (кожного разу після додавання нової функції до компоненту), а також модульні тести легко піддаються автоматизації.

Критерій проходження тестів: всі модульні тести проходять успішно, тобто всі основні функціональні можливості компонентів працюють як розраховувалось. Відсоток загальної кількості успішних тестів має становити більше 97%, і не повинно бути жодних критичних чи серйозних помилок.

Якщо є будь-які критичні чи мажорні помилки в основній функціональності, то тестування користувацького інтерфейсу, зручності повинні бути призупинені. Модульне, інтеграційне та системне тестування можна продовжувати.

Відновлення відбудеться після виправлення зазначених помилок.

Якщо з оновленням операційної системи в ній виникає помилка, що заважає нормальному функціонуванню основних елементів гри, тестування для поточної версії повністю зупиняється.

Відновлення станеться, коли розробник ОС виправить помилку.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування відбуватиметься відповідно до функціональних вимог та прописаних сценаріїв використання. Тестувальник вивчає сценарії, на основі них складає тест-кейси, формує очікуваний результат від певних дій. Задача тестування: перевірити, чи відповідають реальні результати очікуваним.

Рекомендоване складання таких груп тест-кейсів: негативні, позитивні, специфічні та додаткові.

До кожного класу, що потребує тестування (зазвичай це viewModel – GameViewModel, LoginViewModel) – створено відповідний файл *назва класу*Tests, де, за допомогою нативного фреймворка XCTest створено клас та функції тестування. Вони автоматично підтягуються схемою IDE та виконуються на запит розробника.

Клас тестування є нащадком класа XCTestCase та має таку структуру:

а) функція setup() – тут відбувається створення моделі, що тестується та її налаштування під потреби тестування. Ця функція викликається перед початком виконання усіх тестів;

б) функція tearDown() – викликається після виконання всіх функцій-тестів цього класу. Тут можна проаналізувати результати тестування та «зачистити» певні дані за потреби;

в) власне, функції тестування. Функції розпізнаються IDE як функції тестування, якщо вони містять хоча б один виклик функції XCTAssert або подібних. Ця група функцій створена для порівняння очікуваних від системи результатів з реальними. Якщо всі XCTAssert повернули результат «Істина», тест вважається пройденим. Якщо хоч одна функція тестування з усього проекту повернула результат «Хибність», тестування вважається «проваленим», і проект у такому стані не може піти до користувача, доки проблема не буде вирішена.

Інші види тестування проводяться мануально. Тест-кейси для перевірки:

- вхід через Facebook (таблиця 4.1);
- почати гру (таблиця 4.2);
- пауза (таблиця 4.3);

– атака (таблиця 4.4).

Таблиця 4.1 – Перевірка можливості входу через Facebook

Мета тесту	Перевірка можливості входу через Facebook
Початковий стан	Відкрито додаток.
Вхідні дані	Відсутні.
Схема проведення тесту	Натиснути кнопку «Увійти через Facebook», у діалоговому вікні натиснути «ОК».
Очікуваний результат	Відображено екран початку гри з кнопками початку гри та виходу з аккаунту.
Стан програмного продукту після проведення випробувань	Користувача авторизовано в додатку.

Таблиця 3.2 – Перевірка можливості початку гри

Мета тесту	Перевірка можливості початку гри
Початковий стан	Користувача авторизовано.
Вхідні дані	Особисті дані користувача.
Схема проведення тесту	Натиснути кнопку «Встати в чергу»; дочекатись повідомлення, що необхідна кількість гравців набрана та гру розпочато.
Очікуваний результат	Відображено екран гри: локації всіх користувачів зображені маркерами, колір яких відповідає кольору їх команд, відображено рівень їх сили та кнопку паузи.
Стан програмного продукту після проведення випробувань	Гру розпочато.

Таблиця 3.3 – Перевірка можливості паузи гри

Мета тесту	Перевірка можливості паузи гри
Початковий стан	Гру розпочато.
Вхідні дані	Кількість пауз, що залишились у користувача на сьогодні.
Схема проведення тесту	Натиснути кнопку «Пауза».
Очікуваний результат	Відображено екран паузи: кнопки повернутися до гри та вийти з гри.
Стан програмного продукту після проведення випробувань	Для даного користувача гру поставлено на паузу; його локацію більше не бачать інші користувачі, він також не бачить локацію інших.

Таблиця 3.4 – Перевірка можливості атаки

Мета тесту	Перевірка можливості атаки
Початковий стан	Гру розпочато.
Вхідні дані	Локація клієнта-нападника та клієнта-жертви.
Схема проведення тесту	Підійти до супротивника з меншим рівнем сили на відстань 50 метрів або менше; знаходитись у цій зоні протягом 5 хвилин, доки не надійде повідомлення про успішність атаки.
Очікуваний результат	Гравця-жертву більше не відображено на карті, гравцю-нападнику нараховано 12 балів сили.
Стан програмного продукту після проведення випробувань	Гравець-жертва вибув з гри, гравцю-нападнику нараховано бали сили.

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ О.А. Павлов
“___” _____ 2019 р.

Мережеве ігрове програмне забезпечення з використанням геолокації

Керівництво користувача

КПІ.ІП-5120.045490.05.34

“ПОГОДЖЕНО”
Керівник проекту:
_____ Г.В. Ісаченко

Нормоконтроль:
_____ М.М. Головченко

Виконавець:
_____А.С. Старченко

Київ – 2019 року

ЗМІСТ

1 ІНСТРУКЦІЯ КОРИСТУВАЧА

1.1 Авторизація

Для неавторизованого користувача початковий екран виглядає так, як зображено на рисунку 1.1.

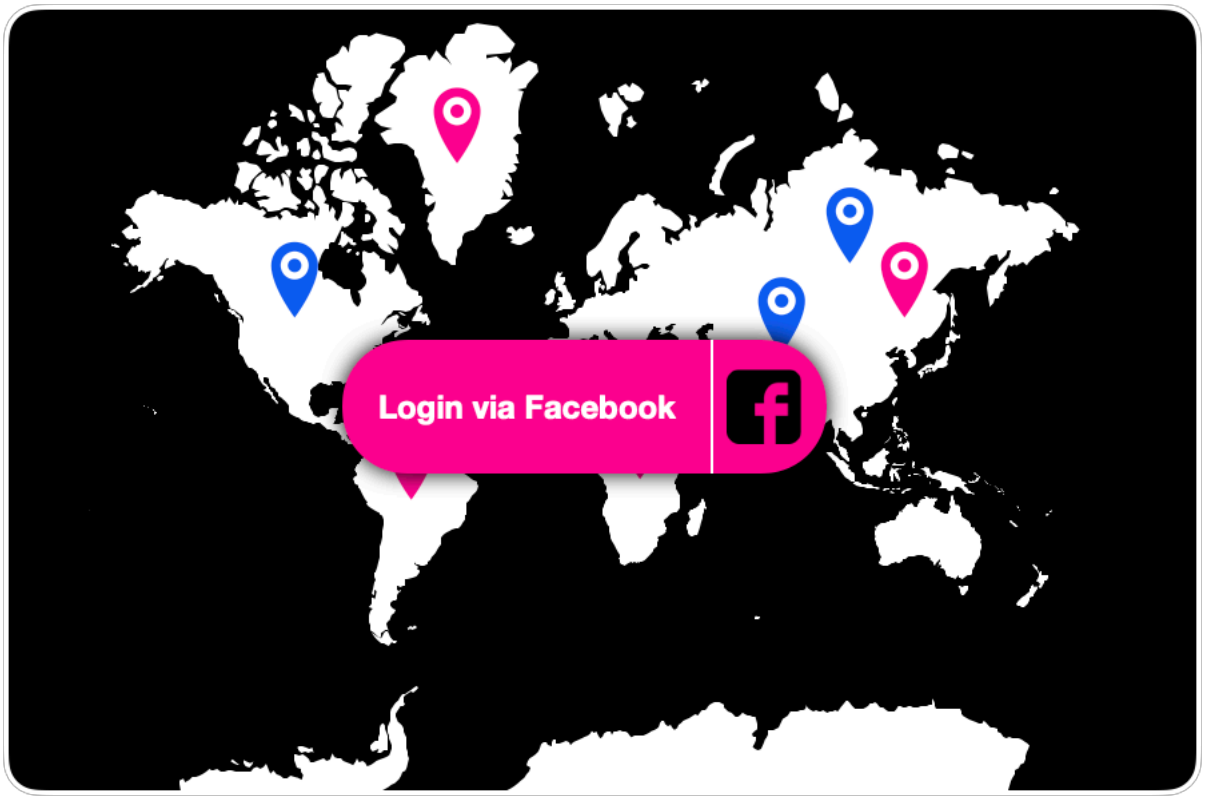

Рисунок 1.1 – Екран авторизації

Він містить єдину кнопку: «Увійти через Facebook». При натисканні на неї відкривається браузер із фейсбук-сторінкою користувача або додаток Facebook (якщо встановлено) з діалоговим вікном.

Для незареєстрованих у грі користувачів з’явиться запитання, чи згодні вони надати свої данні застосунку. Для зареєстрованих також з’явиться це питання з приміткою, що раніше цей користувач уже надавав свої дані цьому застосунку.

В обох випадках треба натиснути “ОК”, користувача поверне до застосунка, де вже буде відображено екран початку гри.

1.2 Початок гри

Екран початку гри зображено на рисунку 1.2.

Рисунок 1.2 – Екран початку гри

На ньому є дві опції: почати гру або вийти з аккаунта.

При виборі опції «Вийти з аккаунта», користувача повертає на екран авторизації.

Якщо вибрати «Почати гру», користувача буде додано в чергу на початок гри, та відобразиться екран черги (рисунок 1.3).

Для гри потрібно 20 людей. На екрані відображається, скільки людей уже набрано.

Також присутня кнопка «Скасувати». Якщо натиснути на неї, з’явиться діалогове вікно з питанням, чи точно користувач хоче скасувати свою участь у грі. Якщо користувач натисне «Так», його буде повернено на екран початку гри, й участь у поточній грі буде скасована.

Коли 20 людей буде набрано, гра автоматично почнеться, та відобразиться власне головний екран гри (рисунок 1.4).

Рисунок 1.3 – Екран черги

Рисунок 1.4 – Головний екран гри

1.3 Ігровий процес

Правила гри наступні:

– є дві команди (сині та червоні);

– у кожного гравця є життєва сила (початковий рівень – 50);

– ціль – «вбити» всіх гравців іншої команди (під «вбити» мається на увазі позбавити життєвої сили повністю);

– щоб «вбити» іншого гравця, треба підійти до нього на радіус 50 метрів або менше, маючи вищій рівень життєвої сили, і пробути в цій зоні не менше 5 хвилин;

– після успішного «вбивства», для жертви гра закінчується, і нападнику нараховується 12 очок;

– під час атаки гравці можуть кооперуватися;

– якщо до нападника приєднуються люди з його команди, час, необхідний на атаку, ділиться на кількість людей, як і бали, що нараховуються у випадку успіху;

– якщо до жертви приєднуються люди з команди, їх рівень життєвої сили сумується, і атака сповільнюється в N разів (де – N, кількість жертв), припиняється взагалі (якщо сумарний рівень сили нападників і жертв – однаковий), або ж жертви стають нападниками (якщо їх сумарний рівень сили перевищив);

– гра закінчується, коли всі гравці однієї з команд вбиті, або через 7 діб.

1.4 Пауза

Кожен день гравець має можливість поставити гру на паузу до 5 разів. Пауза означає, що його локація більше не буде відображатись на карті, а також він не буде бачити локації інших гравців.

Не можна натиснути паузу, коли гравця атакують. Якщо це зробити, зру для гравця буде завершено.

Екран паузи зображено на рисунку 1.5. Він містить кнопку «Повернутися до гри» – повертає у гру, «Вийти з гри» – повертає на екран початку гри, та «Вийти» – повертає на екран авторизації.

Рисунок 1.5 – Екран паузи